Redis as a Reliable Work Queue

Percona University 2015-02-12

Introduction

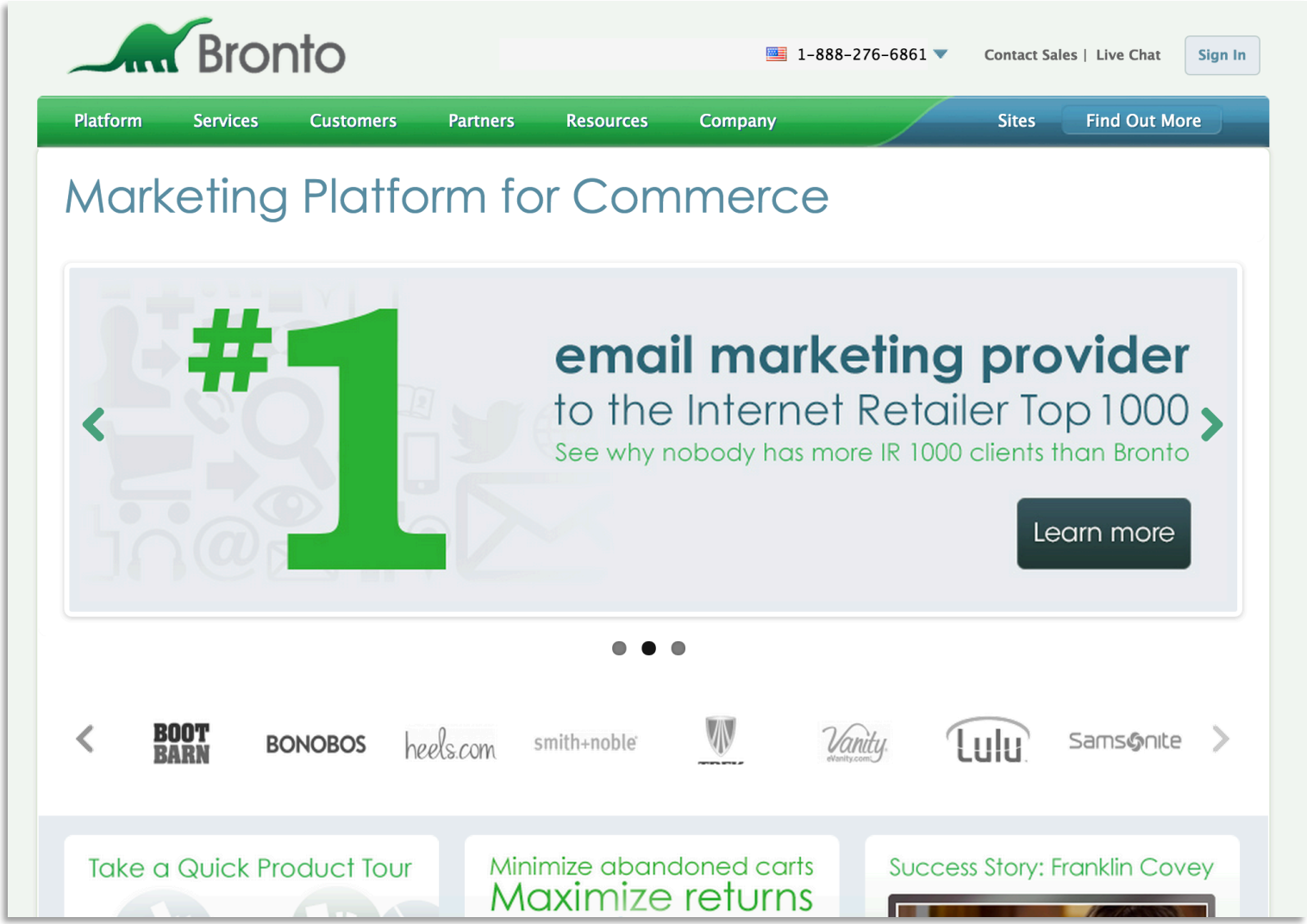
Tom DeWire

Principal Software Engineer
Bronto Software

Chris Thunes

Senior Software Engineer
Bronto Software

Introduction



Introduction

Bronto

Introduction

Bronto Features

- Communication
  - Email
  - Social
  - SMS
- Contact Management
  - Manual
  - Segmentation
- Marketing Automation
  - Workflows
- Commerce Integration
  - Purchase History
  - Cart Recovery
- Integration
  - SOAP/REST API
  - Third Party Connectors

Introduction

Bronto

Cyber Monday 2014

Peak Daily Totals

Emails Sent	Events Processed
~170M per Day	~400M per Day
~2000 per Second	~4700 per Second

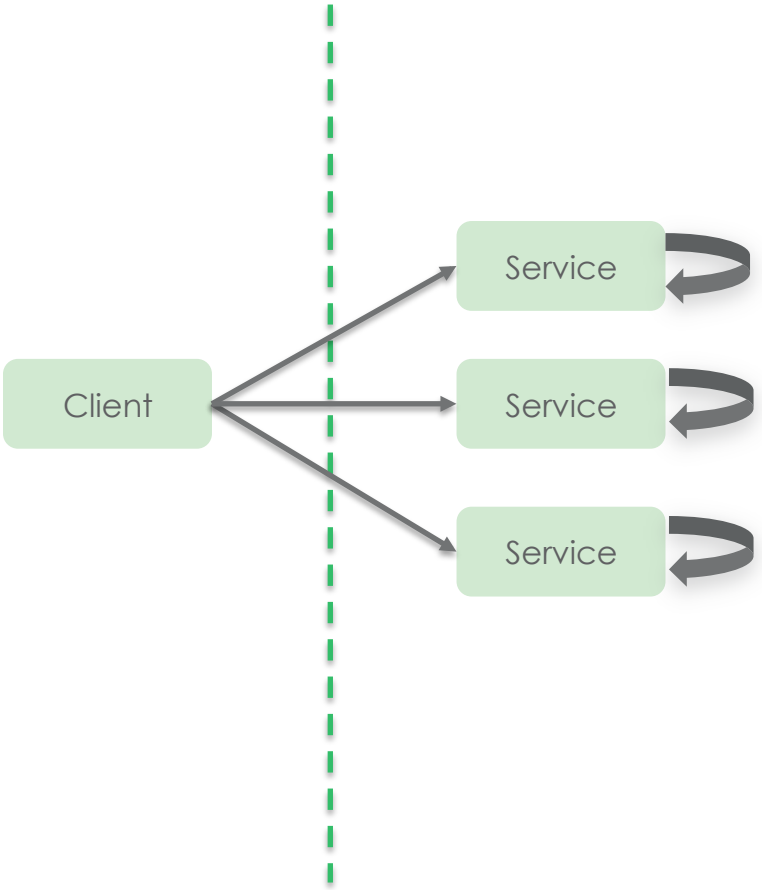
Peak Hourly Totals

Emails Sent	Events Processed
~14M per Hour	~32M per Hour
~3900 per Second	~8900 per Second

Distributed Work Queueing

Bronto

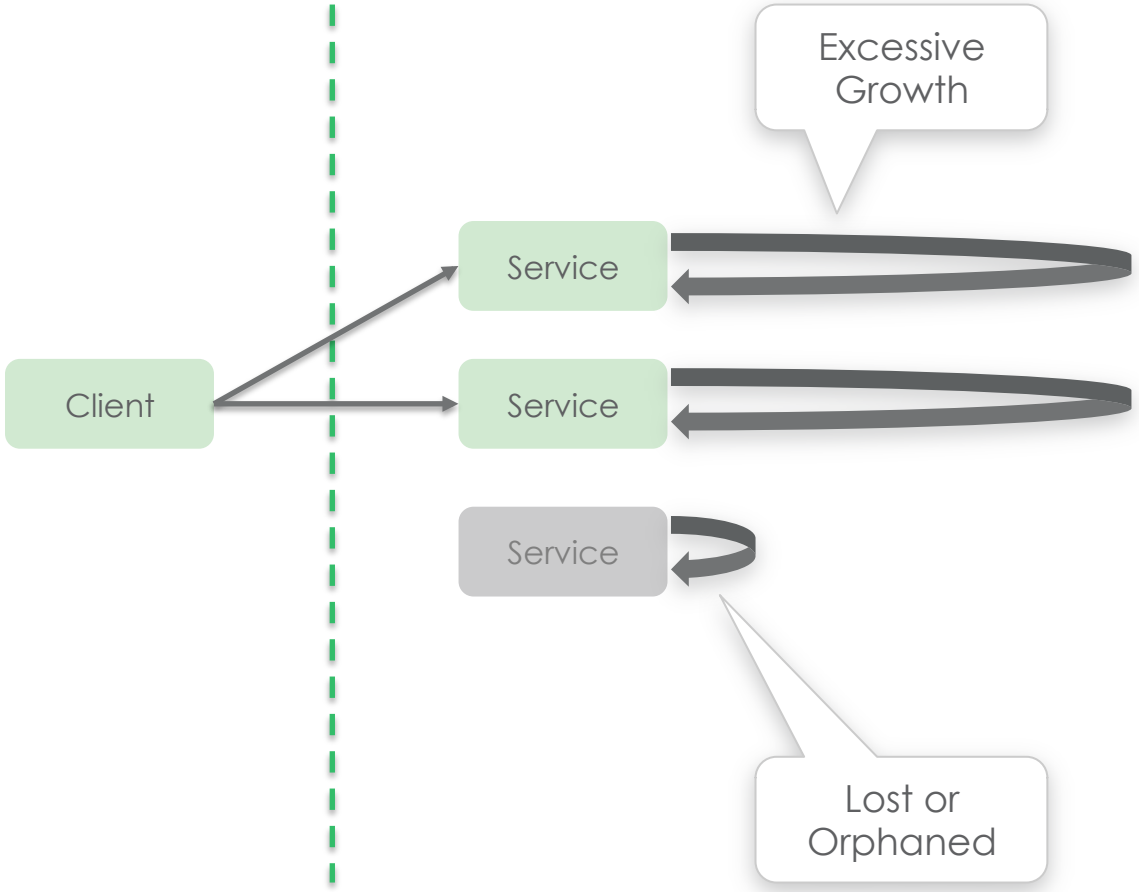
Local work queues…



Distributed Work Queueing

Bronto

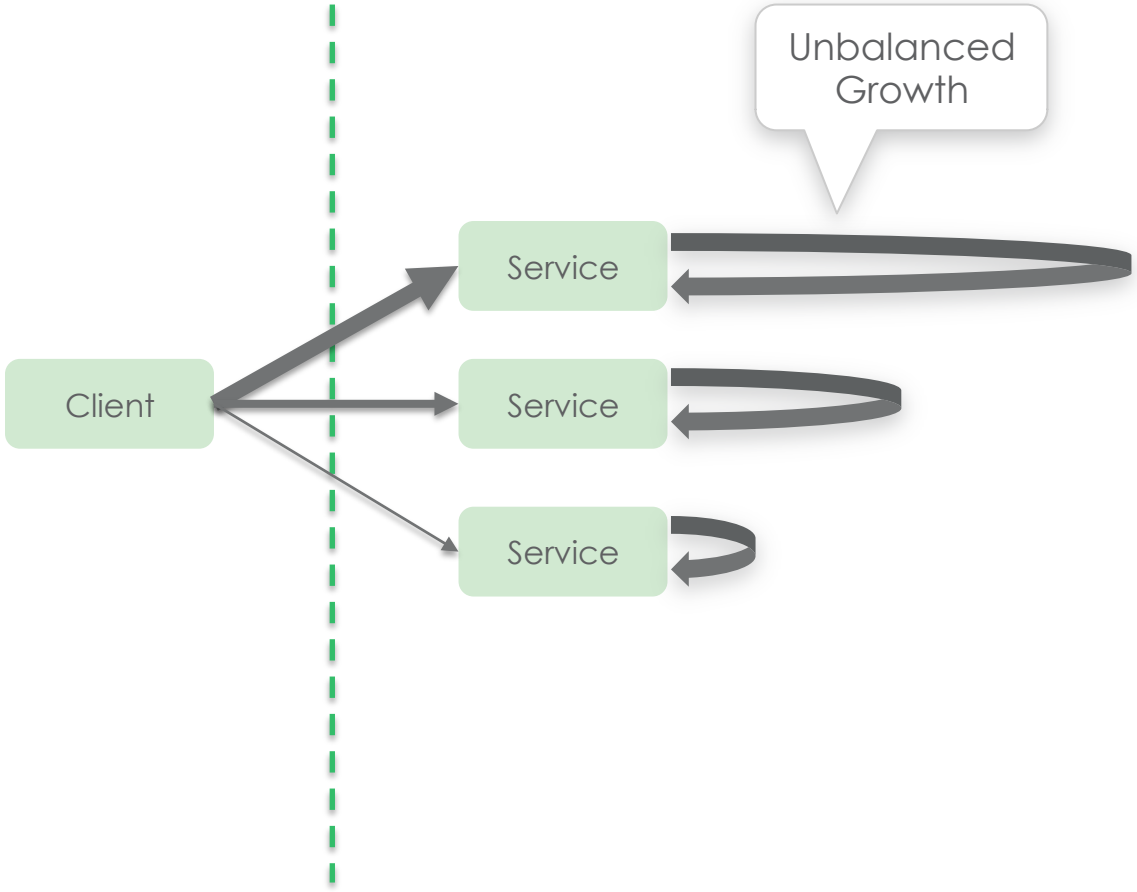
Local work queues…handle failure poorly



Distributed Work Queueing

Bronto

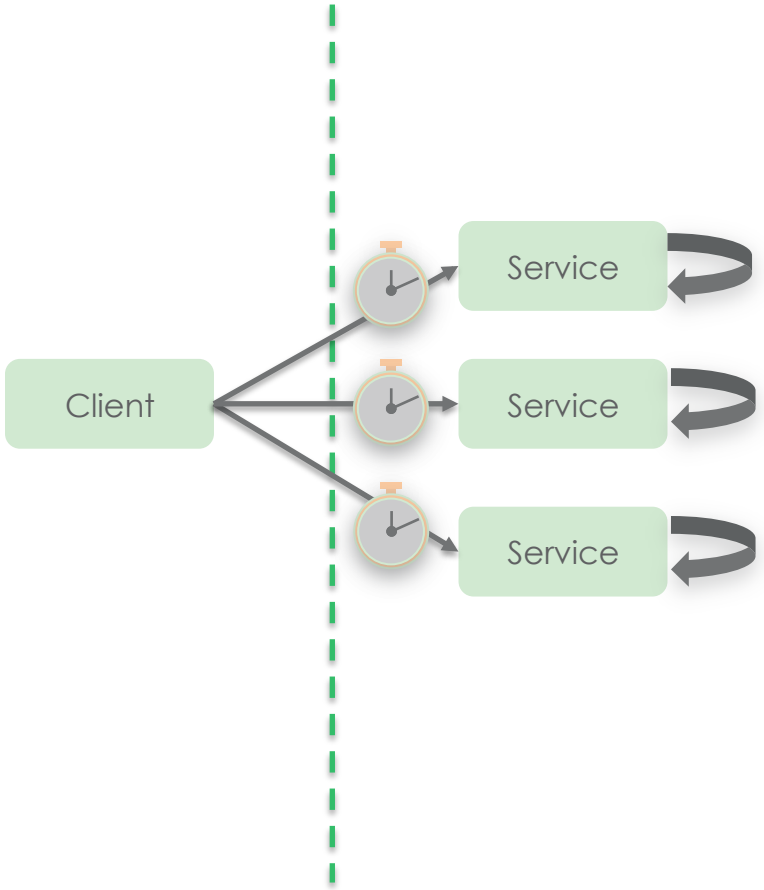
Local work queues…handle unbalanced loads poorly



Distributed Work Queueing

Bronto

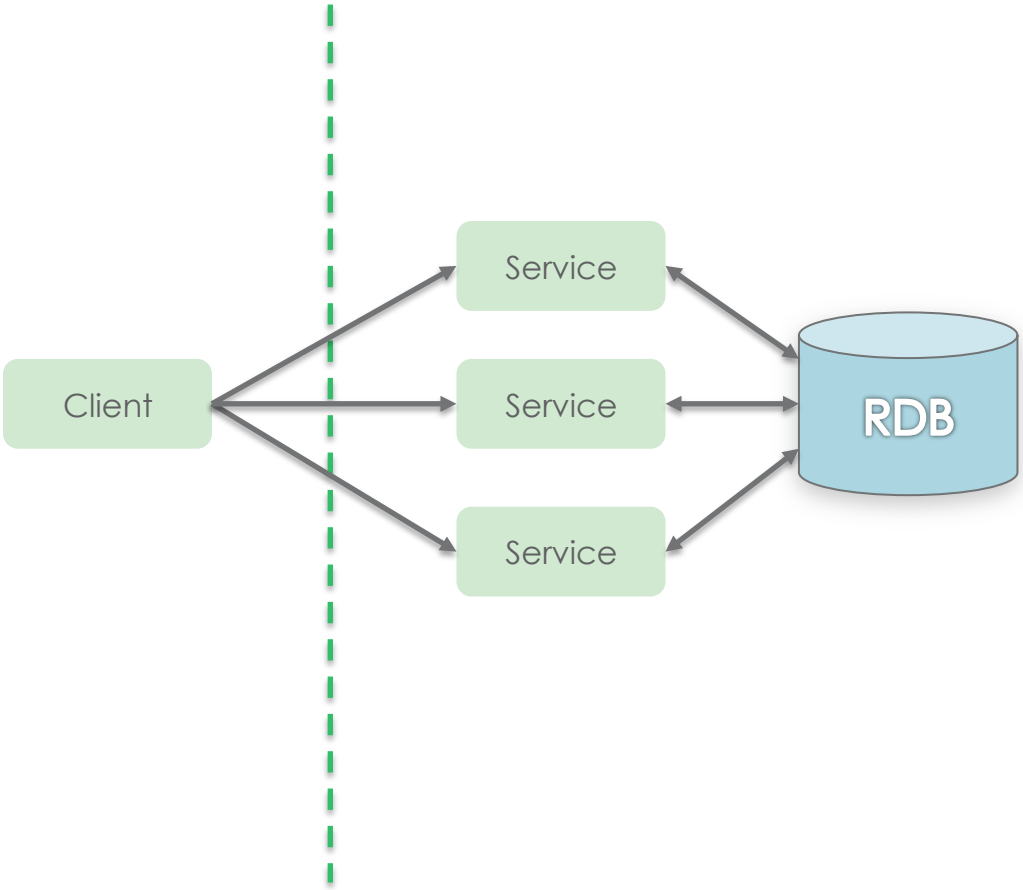
Local work queues… may be bad neighbors



Distributed Work Queueing

Bronto

What about putting the work in a relational database?

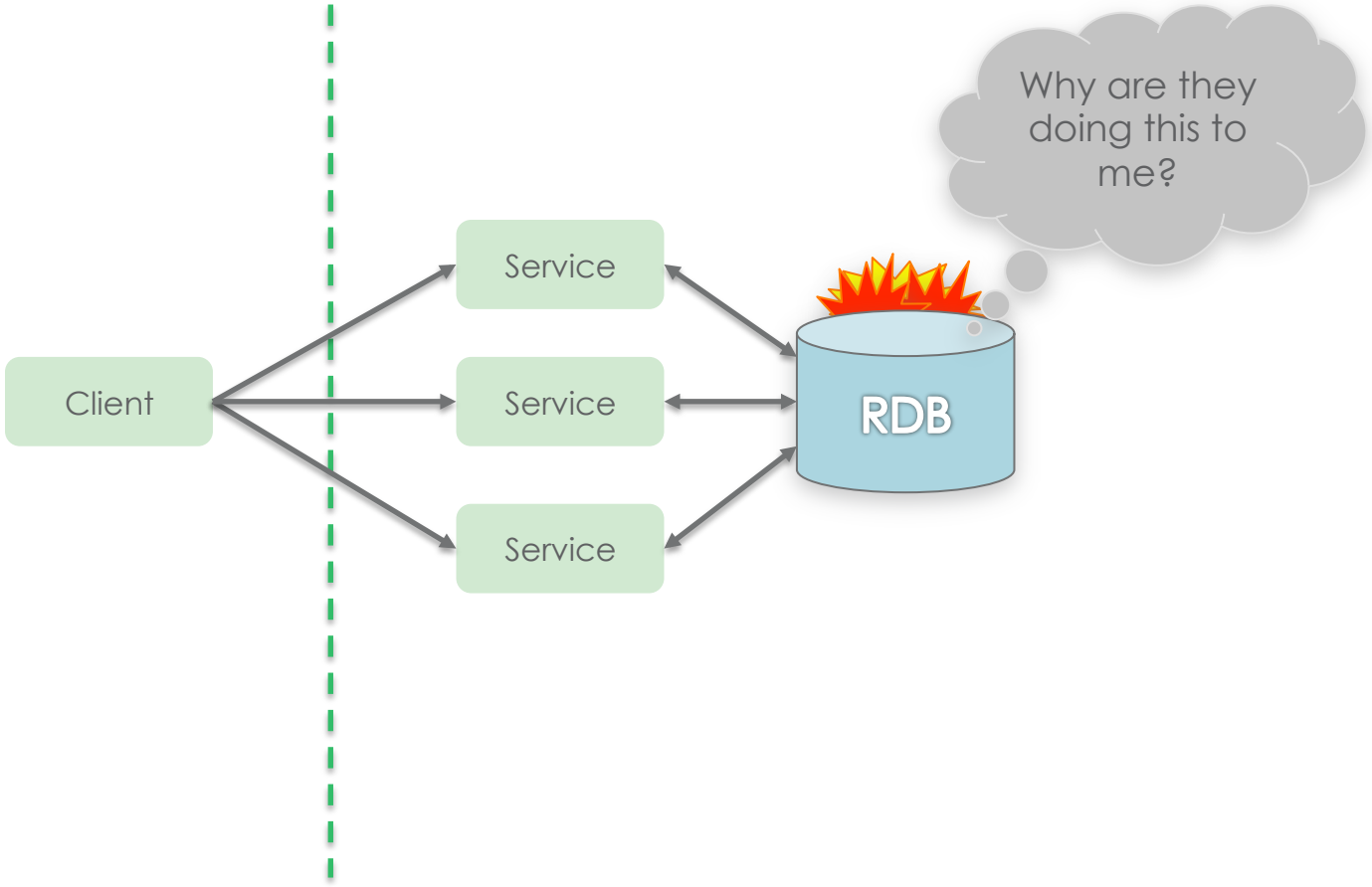


Distributed Work Queueing

Bronto

~~What about putting the work in a relational database?~~

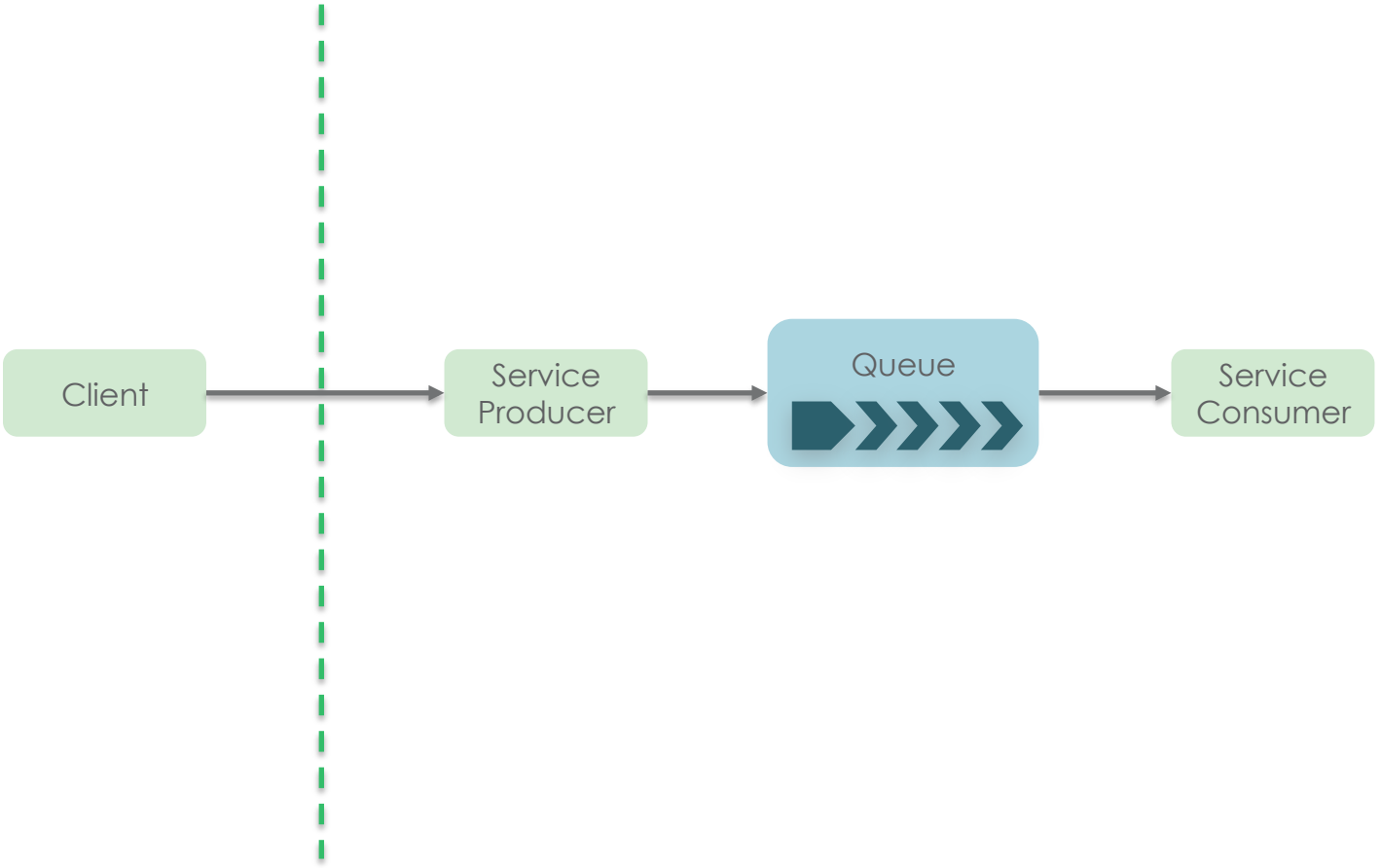
Please don't do that…



Distributed Work Queueing

Bronto

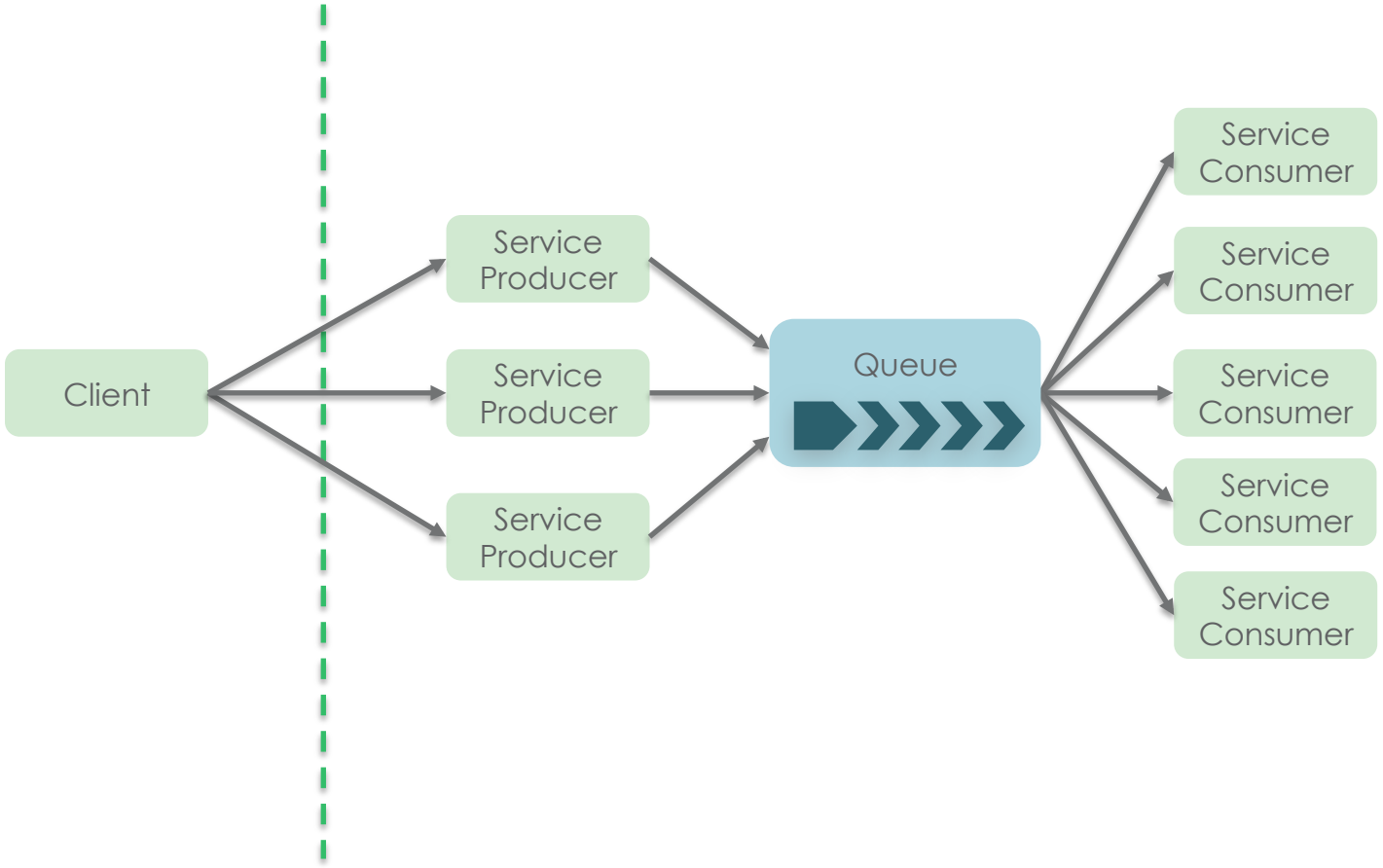
Distributed work queues…



Distributed Work Queueing

Bronto

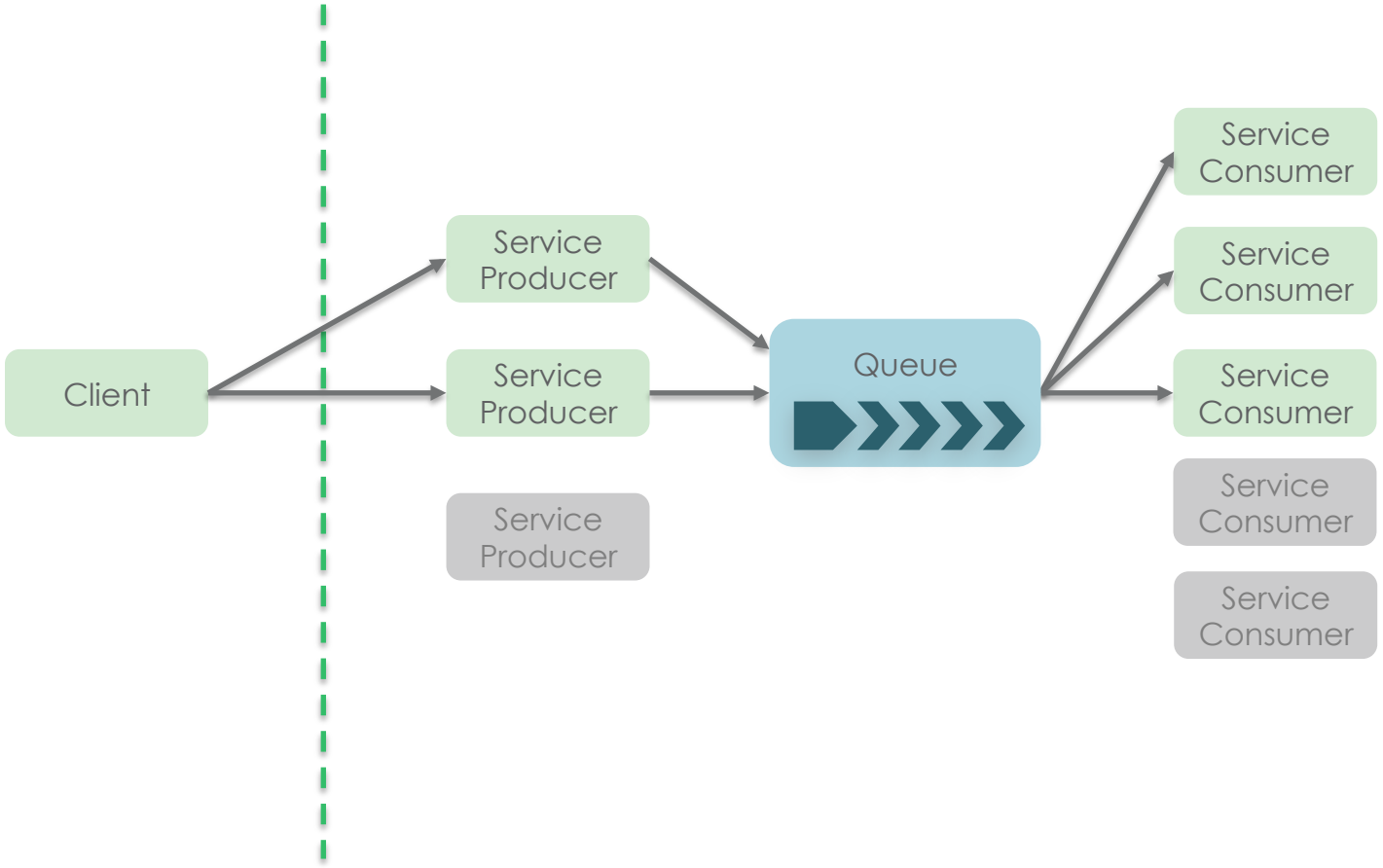
Distributed work queues…decouple producers and consumers



Distributed Work Queueing

Bronto

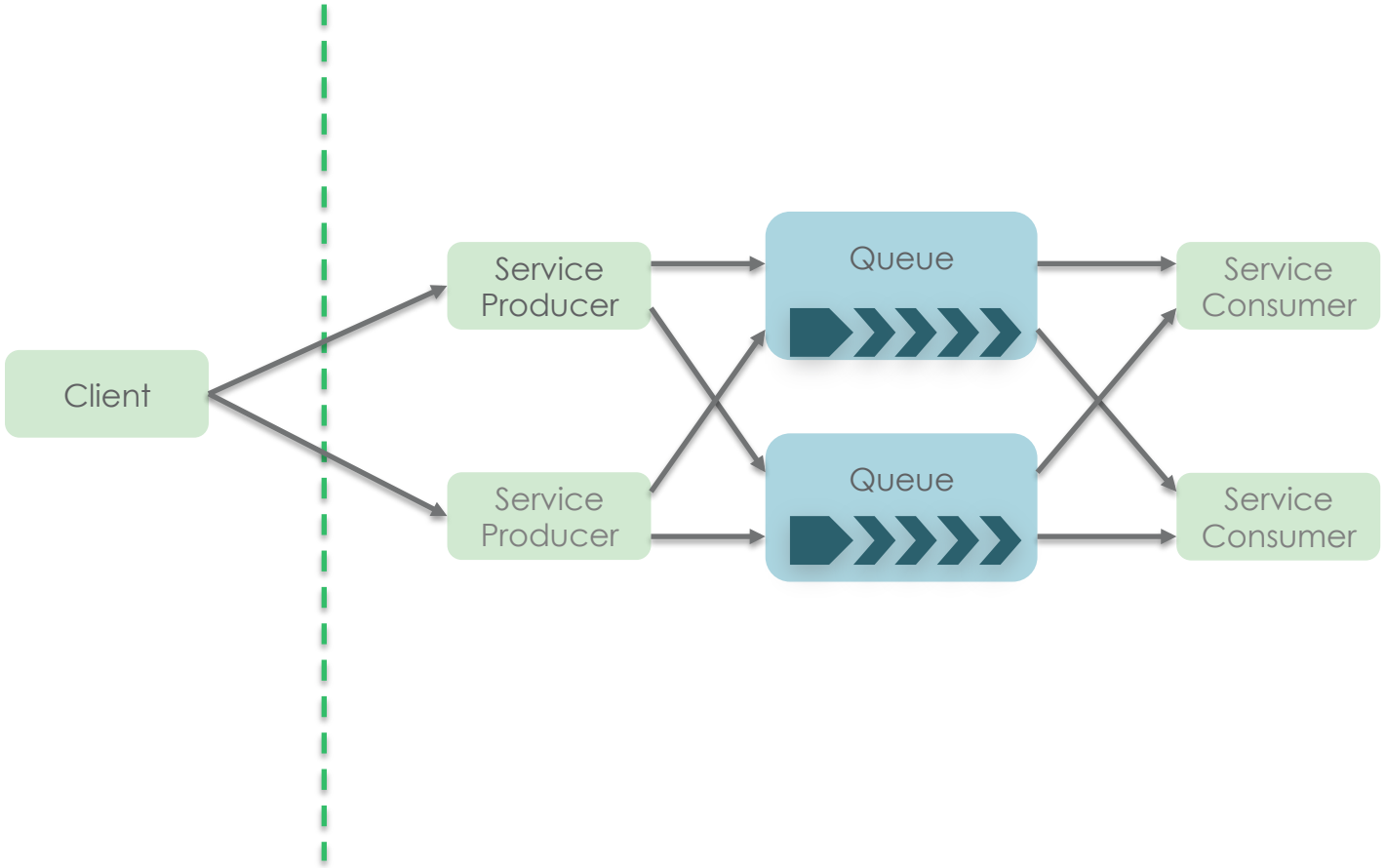
Distributed work queues…are more fault tolerant



Distributed Work Queueing

Bronto

Distributed work queues… partition for availability and/or scale



Distributed Work Queueing

Bronto

Great… But where can I get one?

- Kafka
- Kestrel
- Starling
- BeanstalkD
- SwiftMQ
- RabbitMQ
- ActiveMQ
- Qpid
- Apollo
- SQS (Simple Queue Service)
- …and lots more I've simply forgotten or ignored

There are plenty of options in this space.

Distributed Work Queueing

Bronto

Why did we go with Redis?

- Existing deployment
- Existing operational experience
- Existing development experience
- Works well without specialized hardware
- Favorable balance of throughput vs. durability
- Flexibility to support alternate queue schemes (e.g. with key-based aggregation)

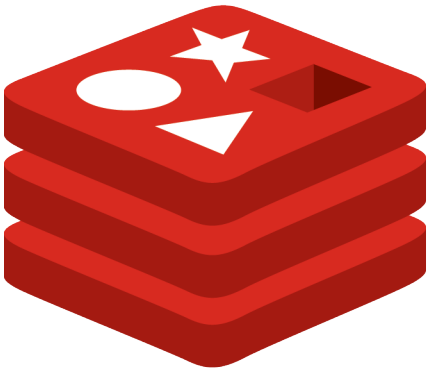
We don't regret building this on Redis, and we feel it will be a solid contribution to the open source ecosystem.

Redis

Bronto

The fundamentals…

- NoSQL database
- Key/Value style
- Data structures
- Memory only
- Durable to disk
- Fast, fast, fast



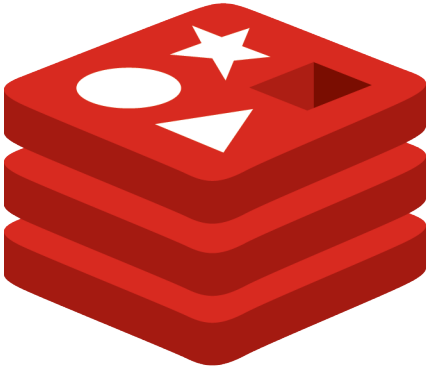
redis

Redis

Bronto

Under the covers…

- C application
- Single threaded
- Strongly consistent
- Durability via snapshots (RDB)
- Durability via redo logs (AOF)
- Scriptable on server-side (LUA)
- Simple protocol
- Replication



redis

Redis

Bronto

Is it ACID? No…

Single Operations

- Atomic
- Consistent
- Isolated

Multi Operation Transactions

- Atomic
- Isolated

Server Scripted Transactions

- Atomic
- Isolated

Redis is not consistent per ACID because it does not support rollbacks.
Redis is not durable per ACID because it does not require persistence to disk.

Bronto

Durability in Redis

RDB (Redis Database)

- Point in time snapshot
- Scheduled or on-demand
- Performed in a forked process
- Compact file format
- Fastest restore time
- Larger window for data loss

There are workloads that can make good use of the scheduled and/or explicit RDB snapshots, but the queue case is not one of them.

Bronto

Durability in Redis

AOF (Append Only File)

- Streaming log of operations
- Periodic log rewriting from live data via fork
- Reduced chance of corruption due to append only strategy
- Multiple fsync() policies
  - Never
  - Every second
  - Every operation
- Slightly reduced performance due to more frequent disk interaction

AOF with the 'every second' fsync policy is a good fit for us.

- No expected data loss due to process failure
- One second of potential data loss due to machine failure

Naïve Queueing

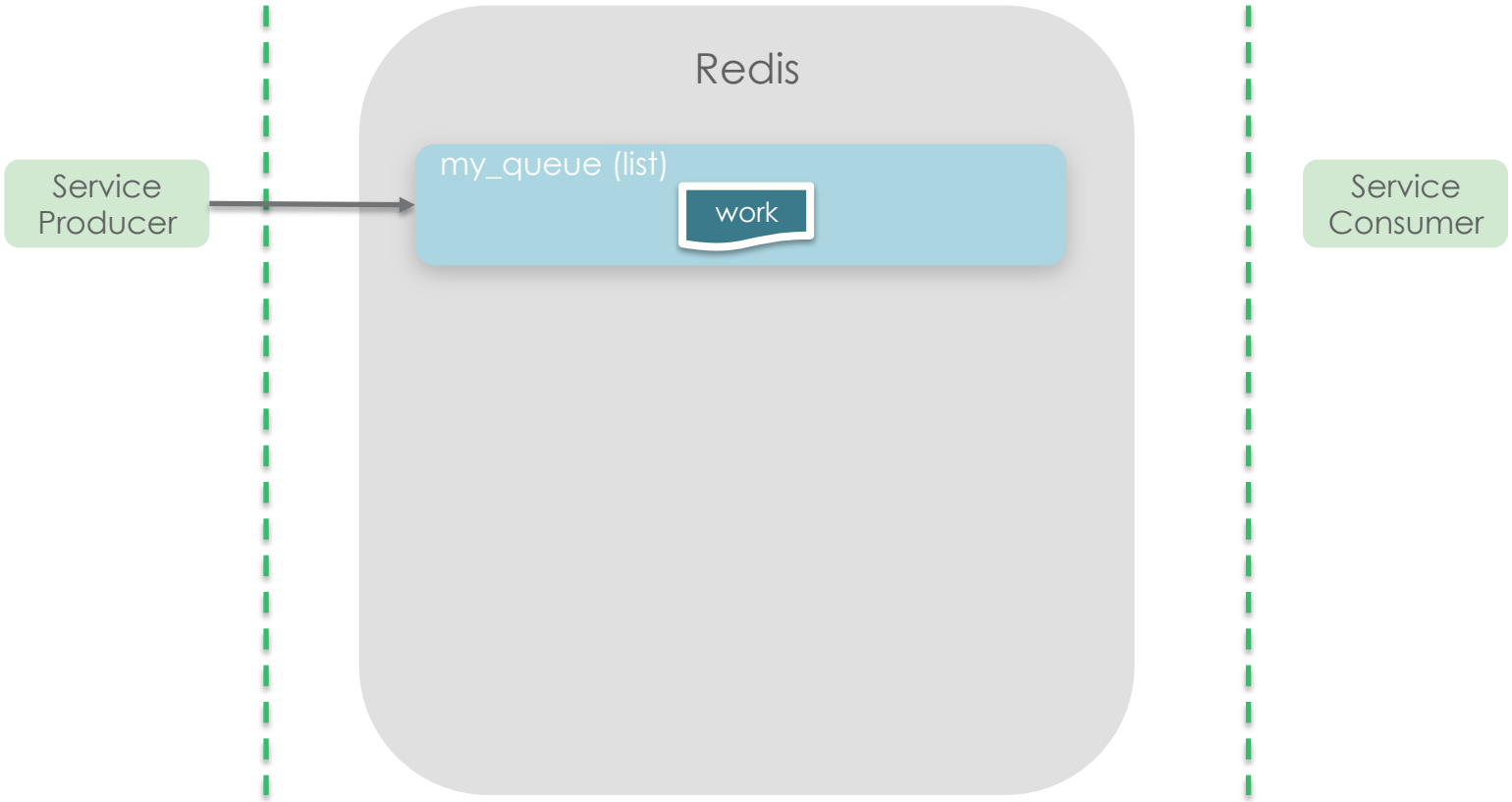
Bronto

Producer creates work payload

Naïve Queueing

Bronto

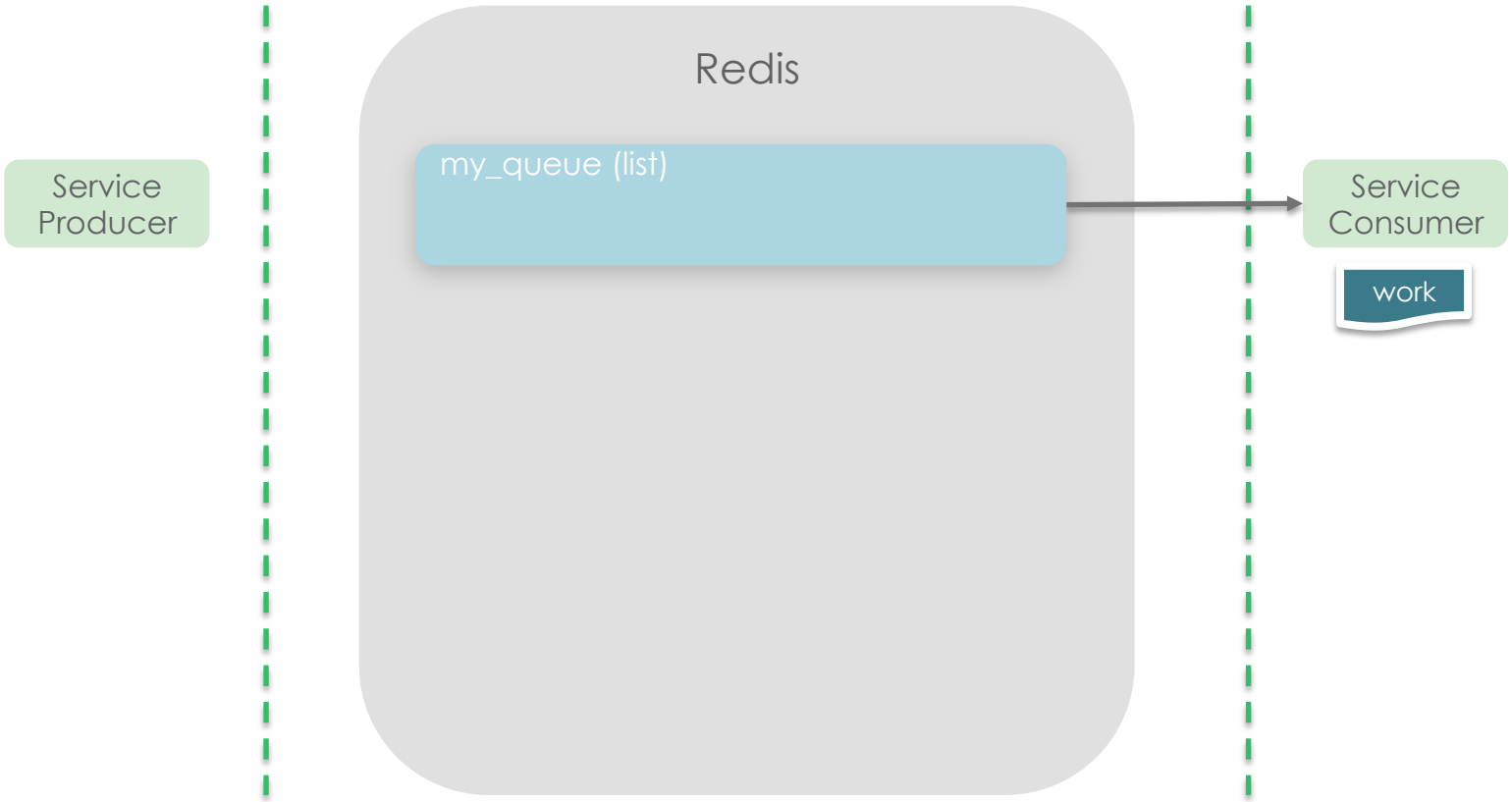
LPUSH my_queue work



Naïve Queueing

Bronto

RPOP my_queue



Naïve Queueing

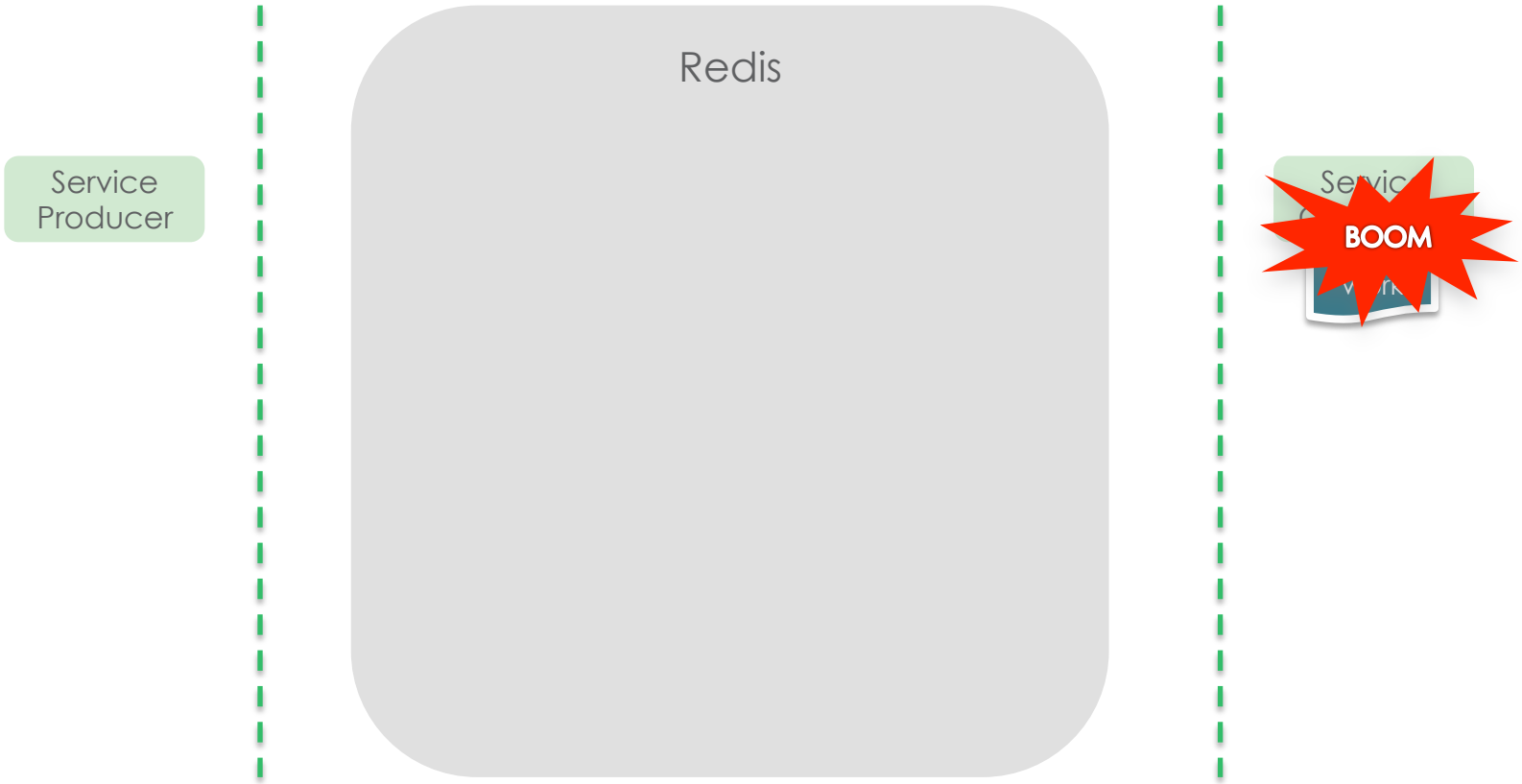
Bronto

Consumer processes work

Naïve Queueing

Bronto

…but what if something goes wrong?

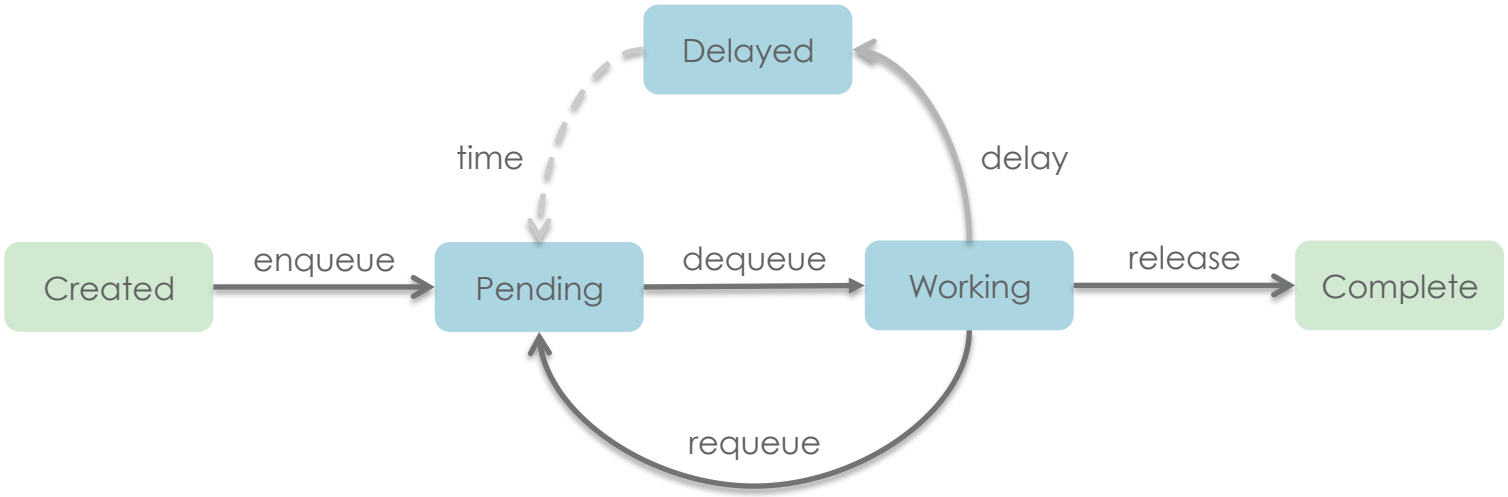


Reliable Queueing

Bronto

Naïve Queueing

Reliable Queueing



Reliable Queueing

Bronto

Redis and LUA Scripting

Think of it as a stored procedure.

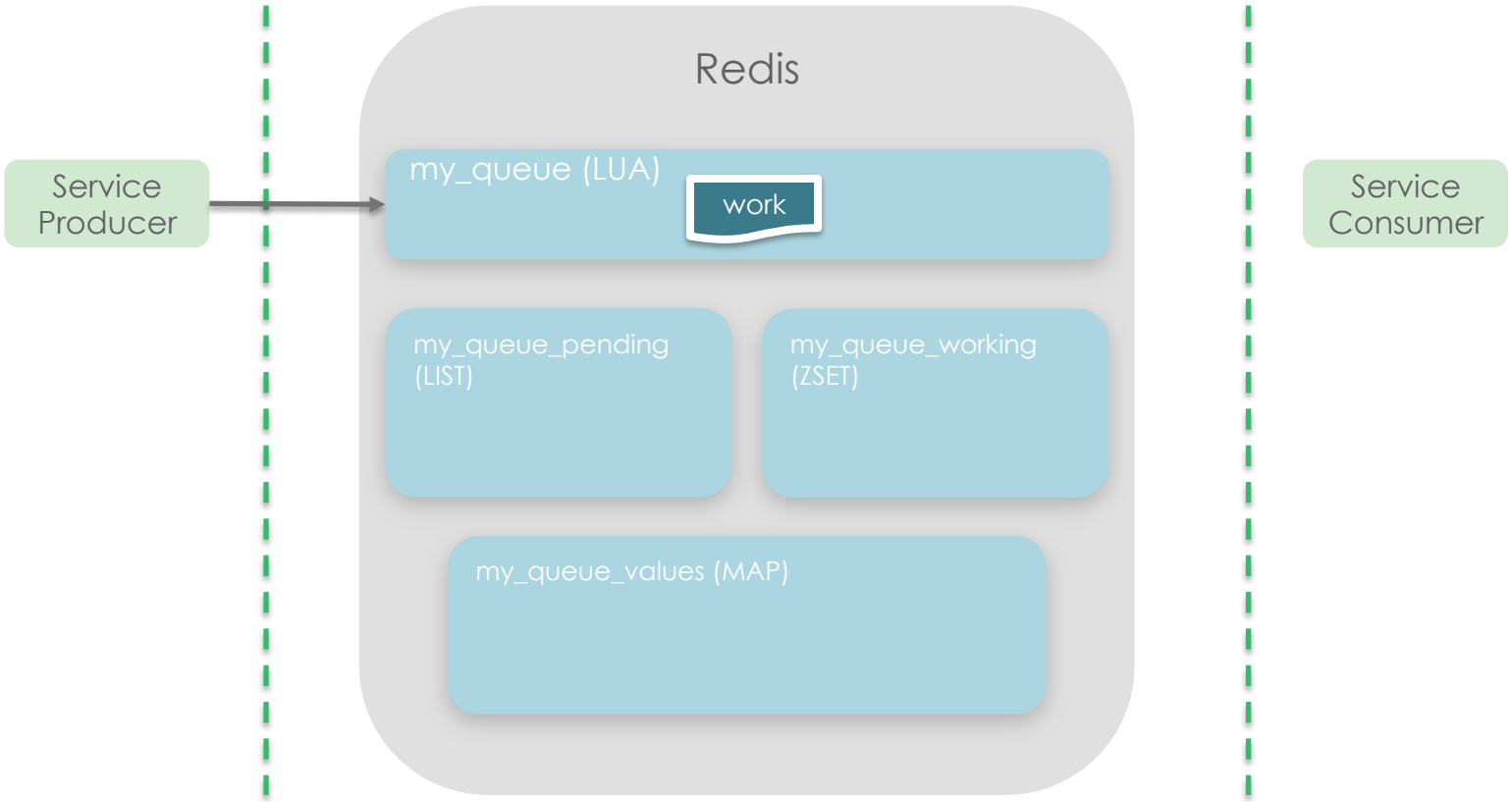
- Loaded via EVAL
- Invoked via EVALSHA
- Atomic execution

```
-- Move the ready UUIDs from the delayed set back into the pending list.
-- These UUIDs are ready when their ZSCORE is less than that of the current time "now",
-- passed in as a parameter to this function.
-- They will be added back to the front of the pending list, rather than the end of it.
local function requeueDelayed(pendingList, delayedZSet, now)
    -- Get the UUIDs of the items ready to be requeued from the delayed set
    local ready_uuids = redis.call('ZRANGEBYSCORE', delayedZSet, 0, now)

    if #ready_uuids == 0 then
        return 0
    end

    -- Move the items from the delayed set to the front of the pending list
    zrem_safe(delayedZSet, ready_uuids)
    rpush_safe(pendingList, ready_uuids)

    return #ready_uuids
end
```

Reliable Queueing

Bronto

enqueue()

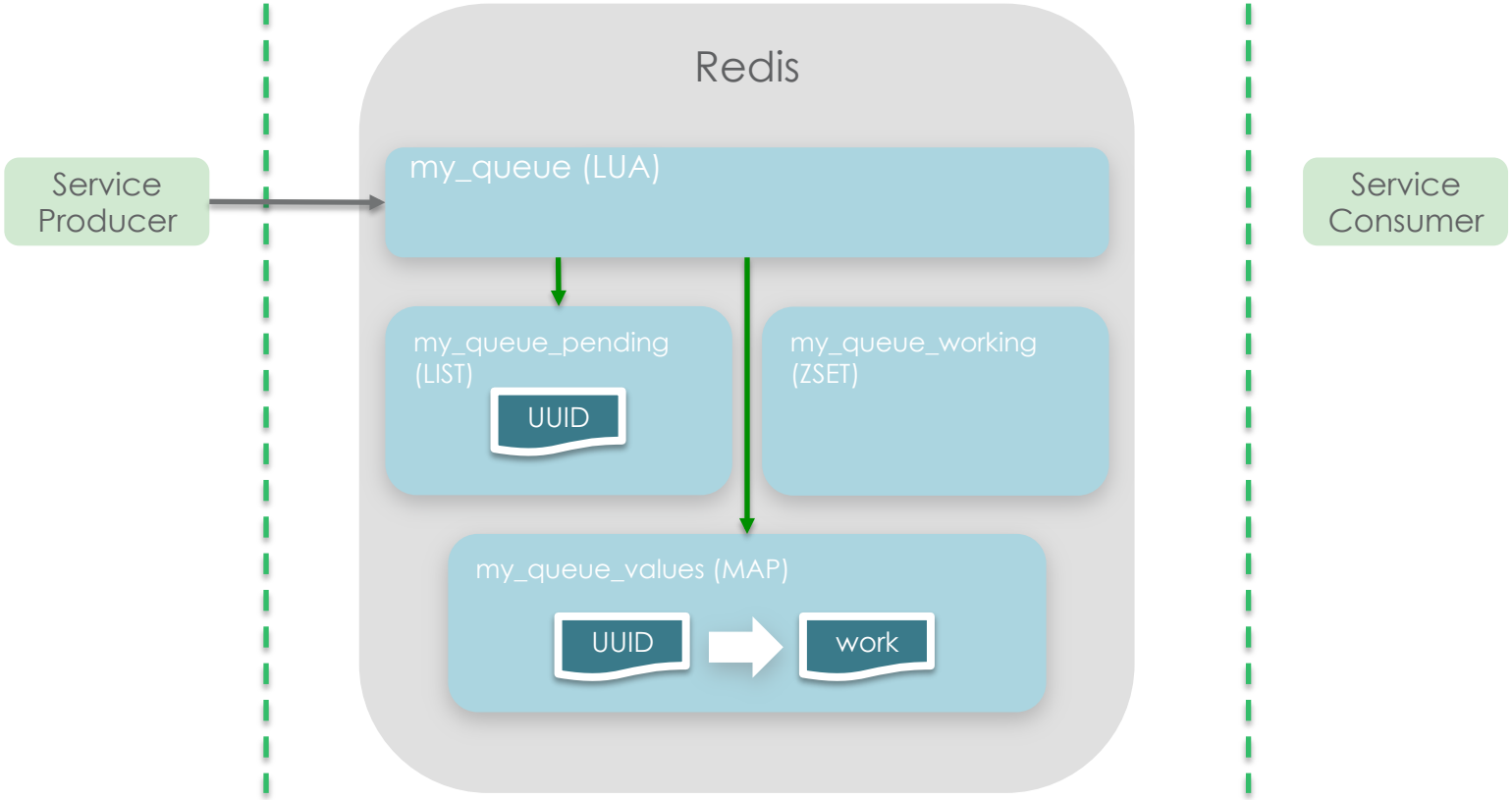


Reliable Queueing

Bronto

enqueue()

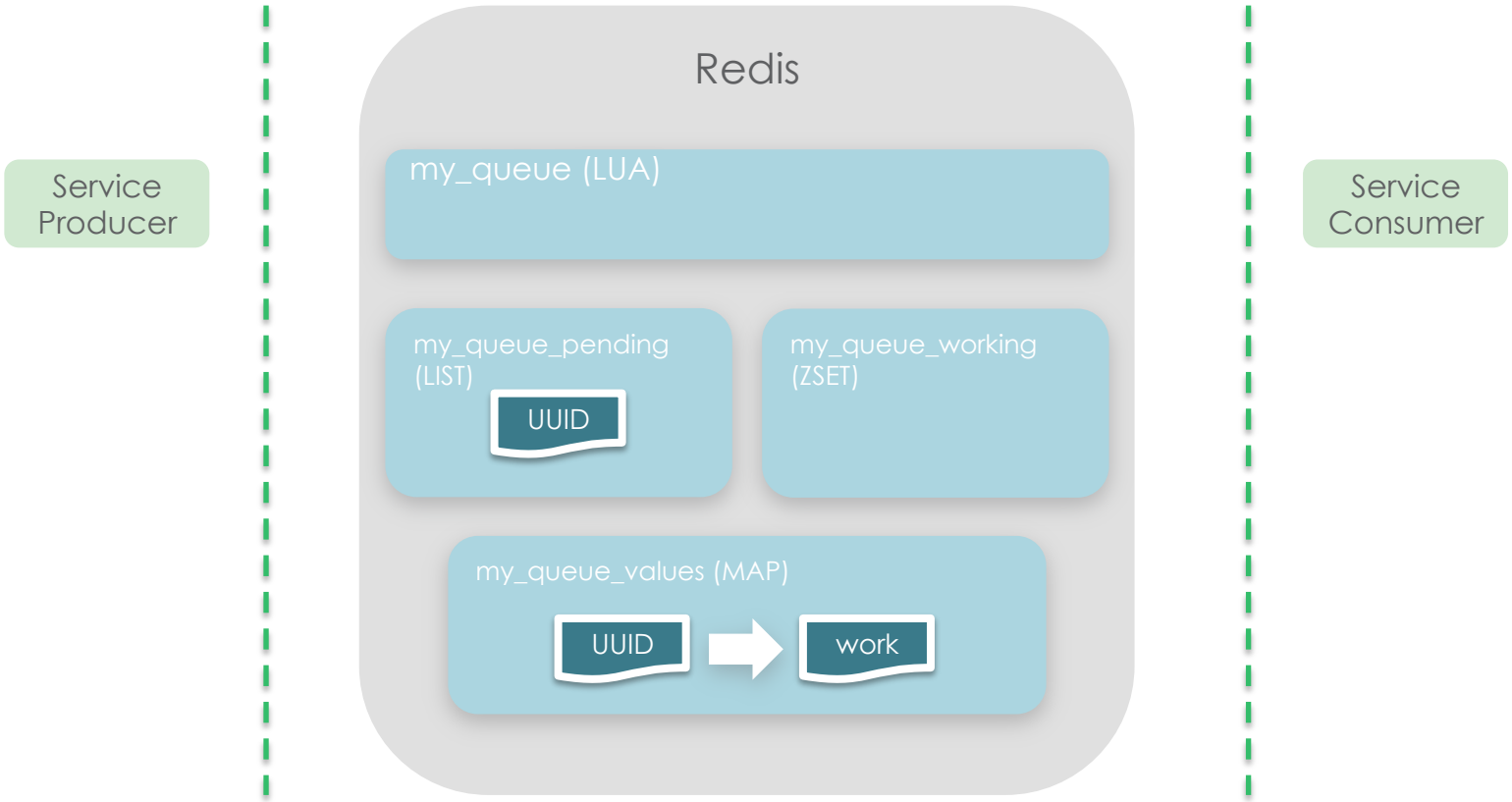
1. Generate {UUID}
2. LPUSH my_queue_pending {UUID}
3. HSET my_queue_values {UUID} {work}



Reliable Queueing

Bronto

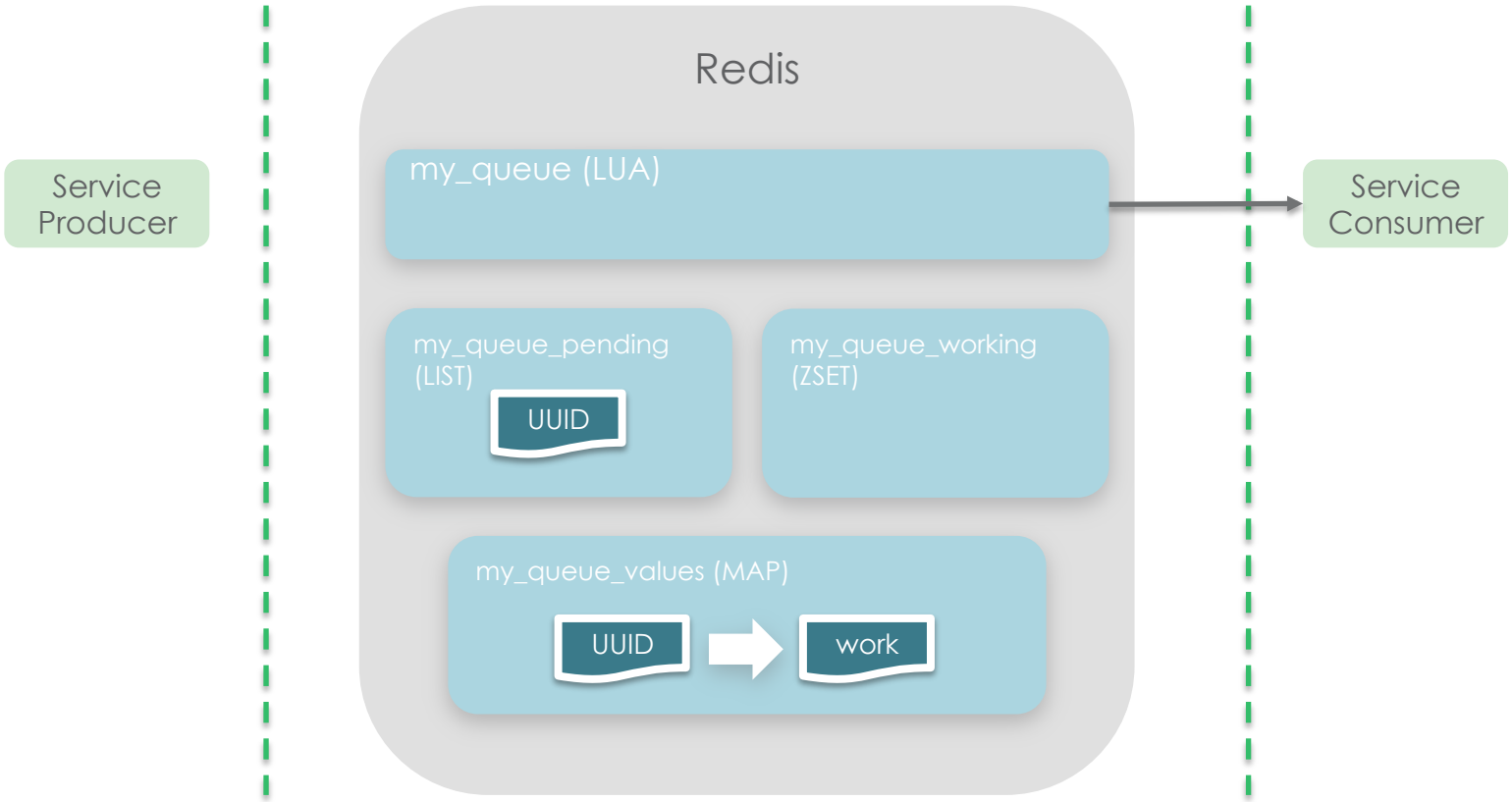
The work is now in the *pending* state.



Reliable Queueing

Bronto

dequeue()

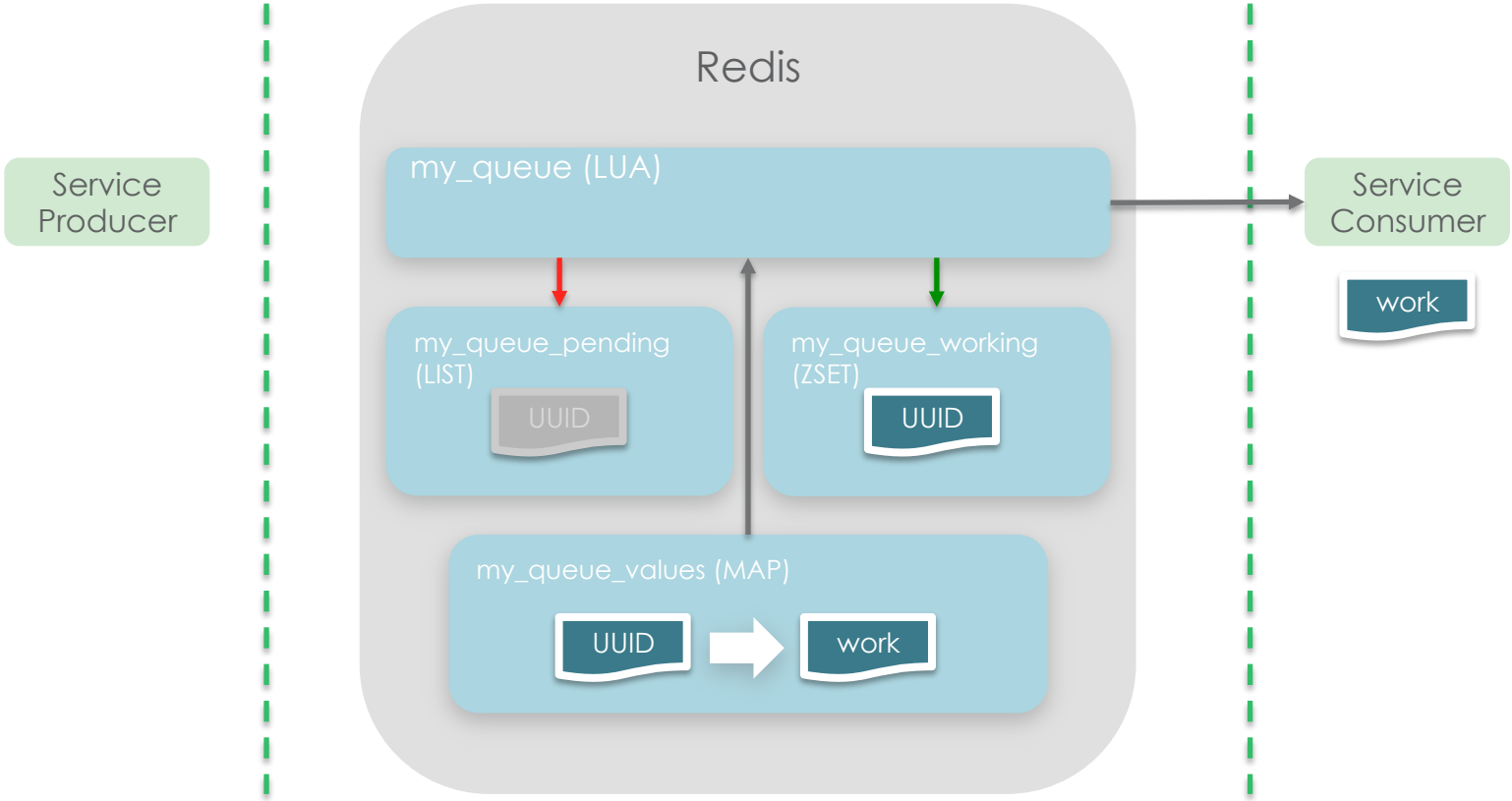


Reliable Queueing

Bronto

dequeue()

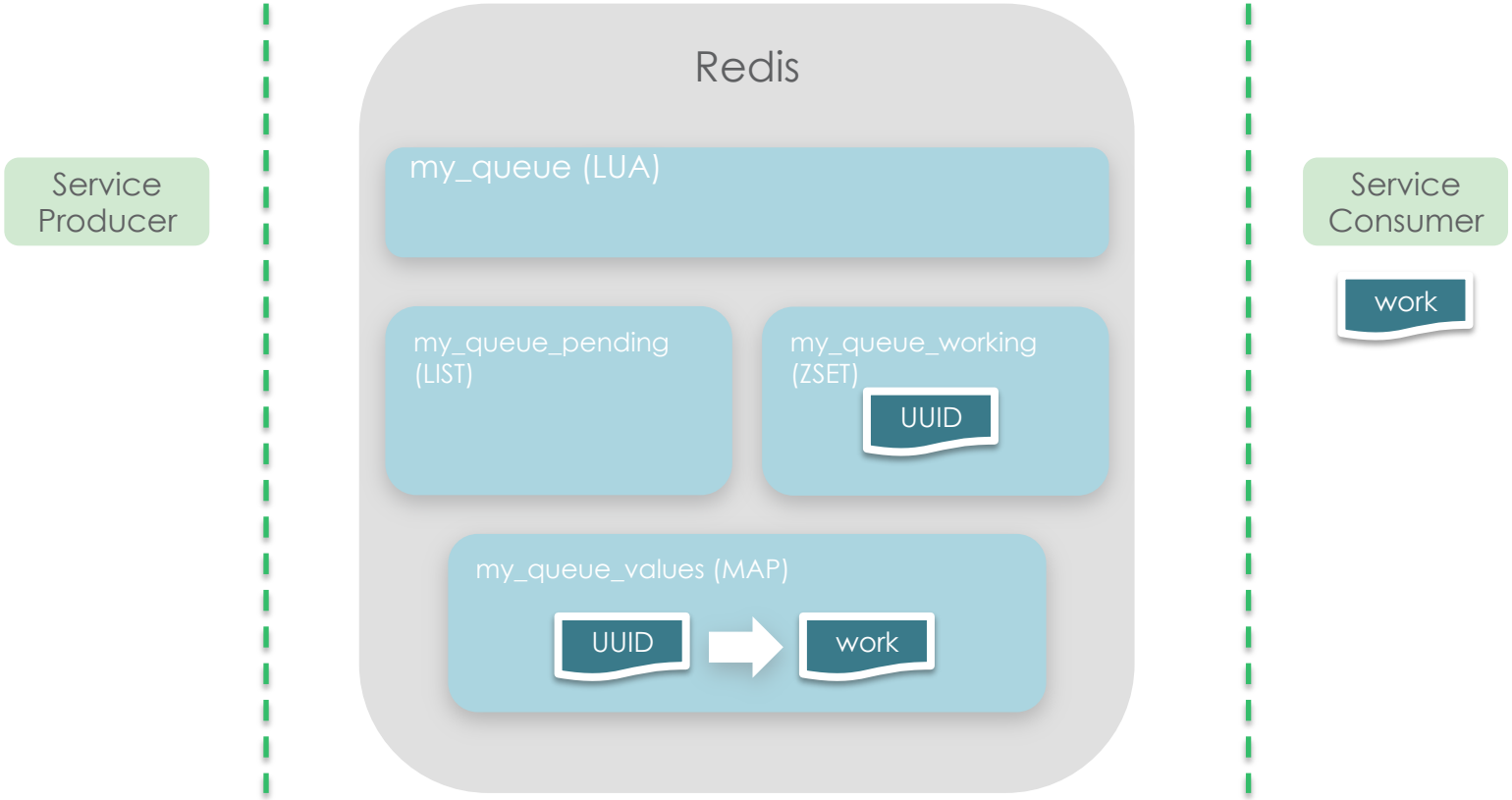
1. RPOP my_queue_pending
2. ZADD my_queue_working {timestamp} {UUID}
3. HGET my_queue_values {UUID}
4. Return {work} to consumer



Reliable Queueing

Bronto

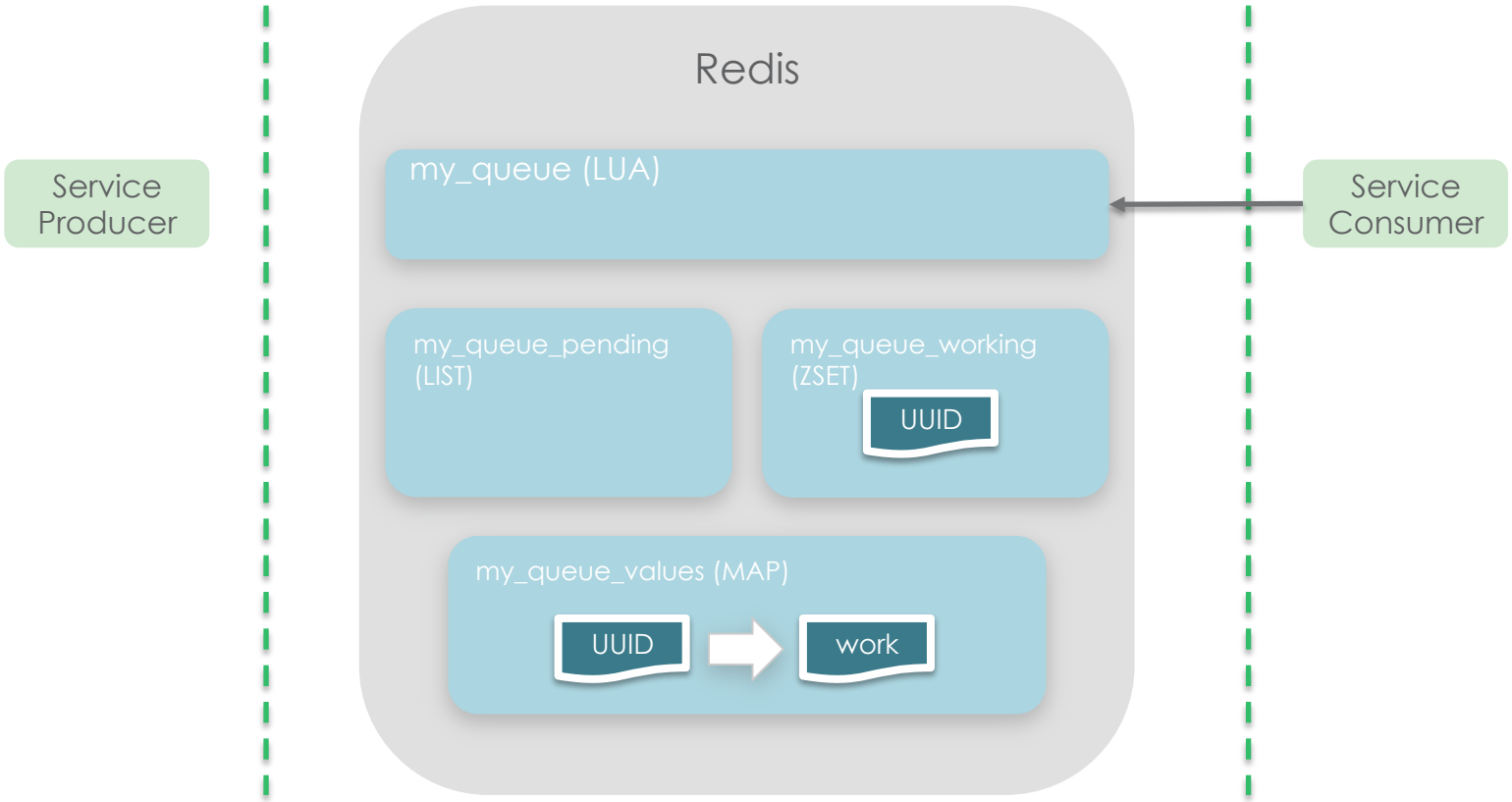
The work is now in the *working* state, safely dequeued, with an immutable copy still on the Redis server.



Reliable Queueing

Bronto

release()

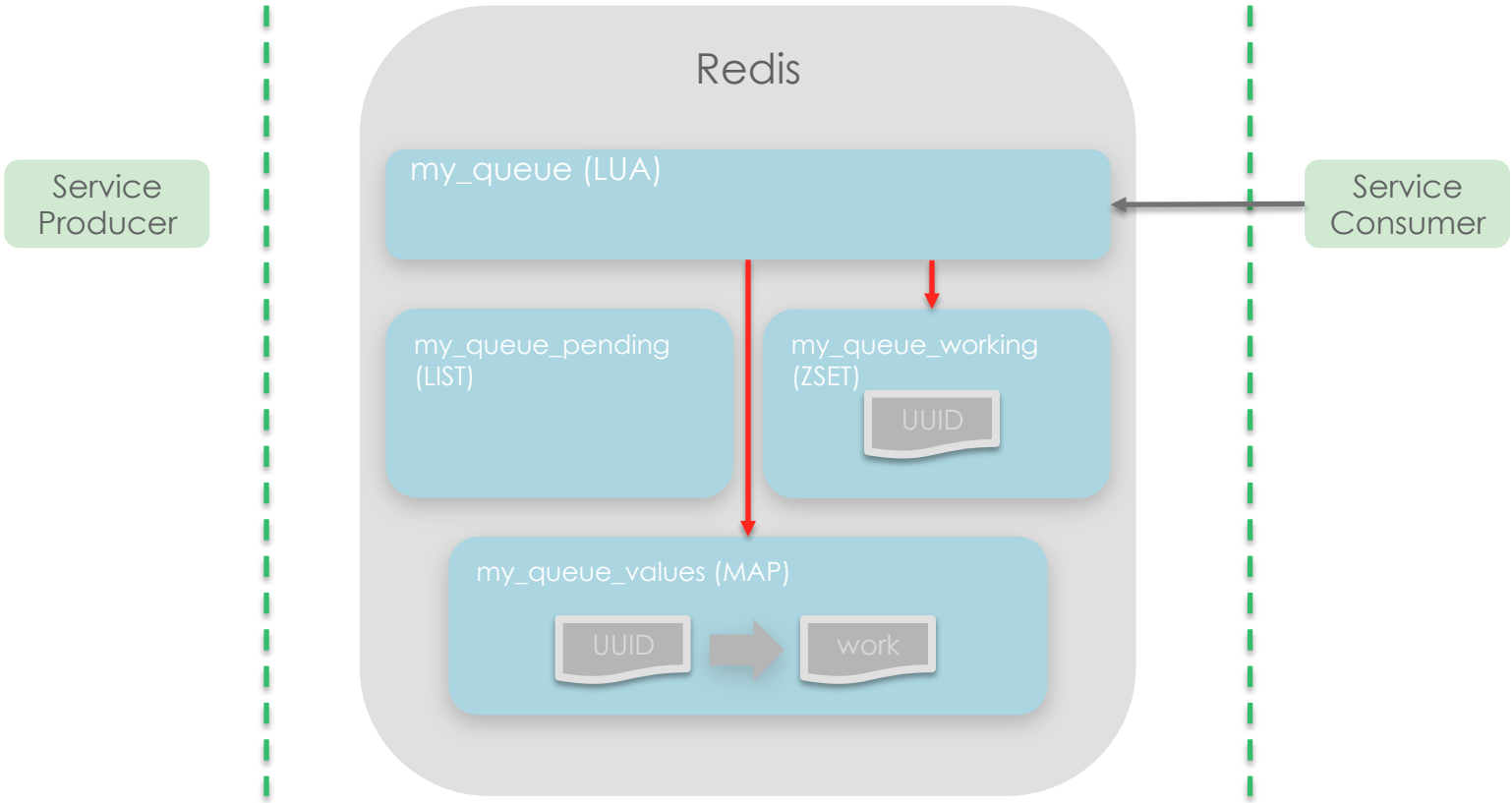


Reliable Queueing

Bronto

release()

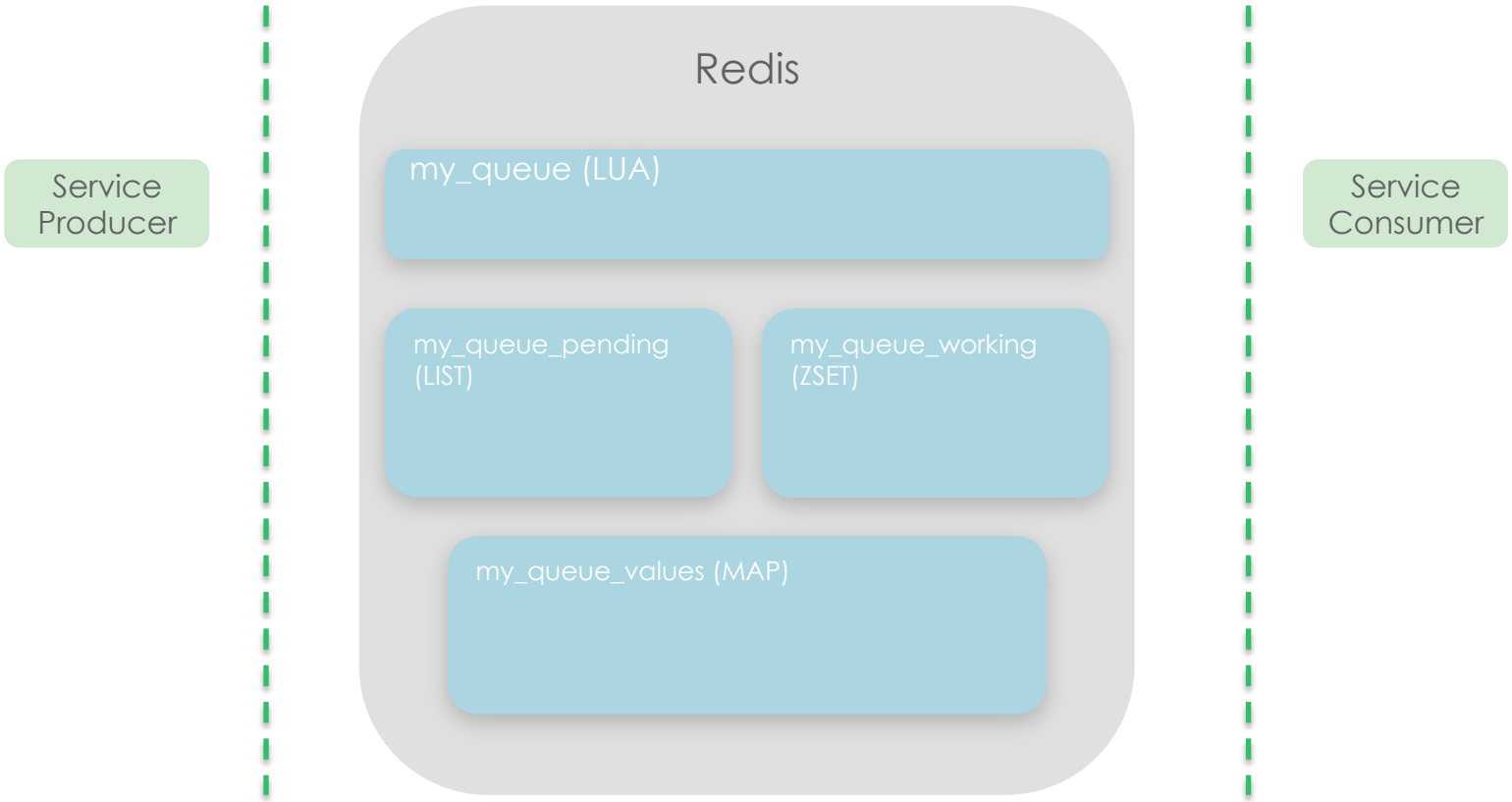
1. ZREM my_queue_working {UUID}
2. HDEL my_queue_values {UUID}



Reliable Queueing

Bronto

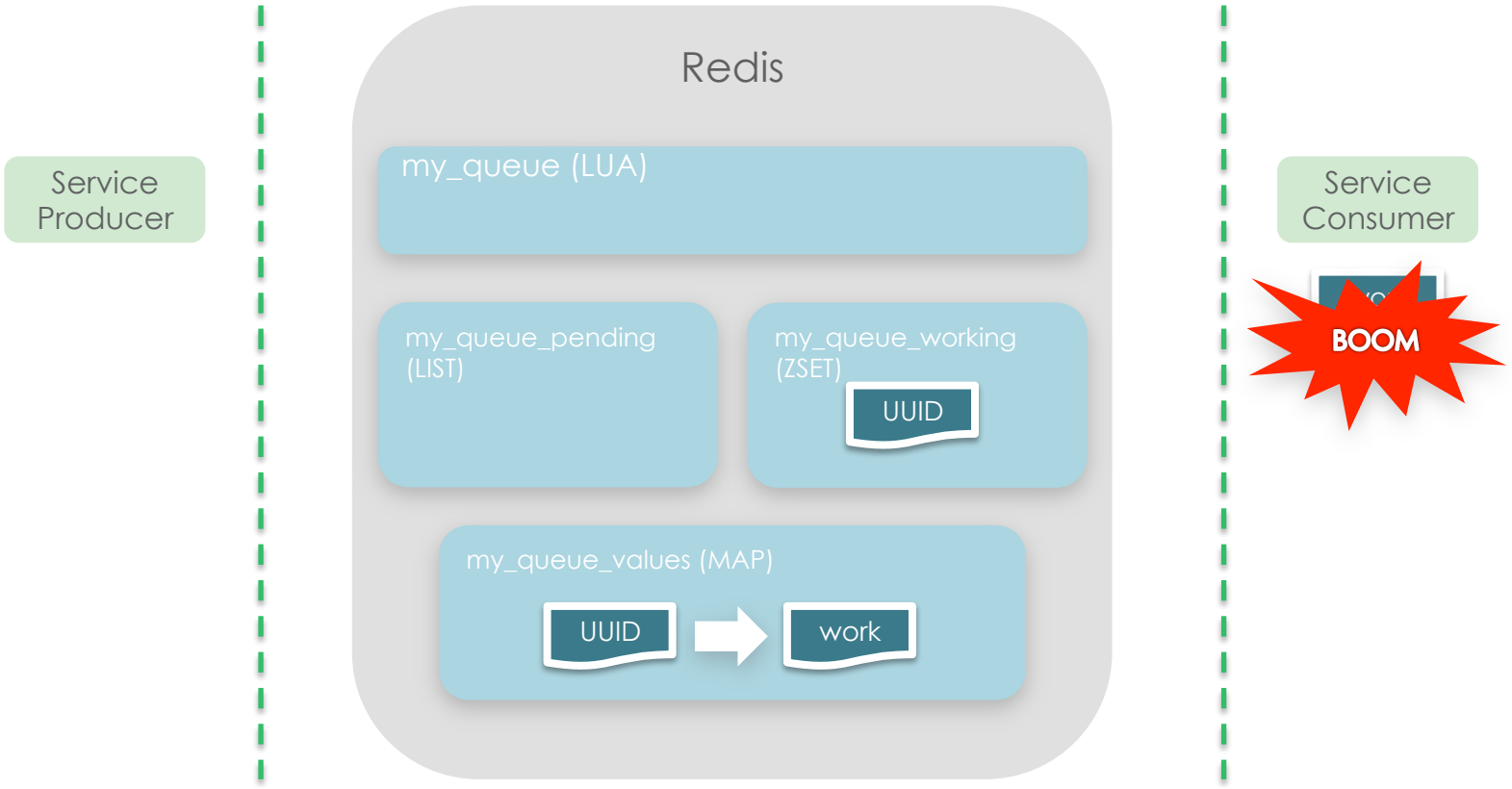
The work is now in the *completed* state, completely processed, with no copy remaining in Redis.



Reliable Queueing

Bronto

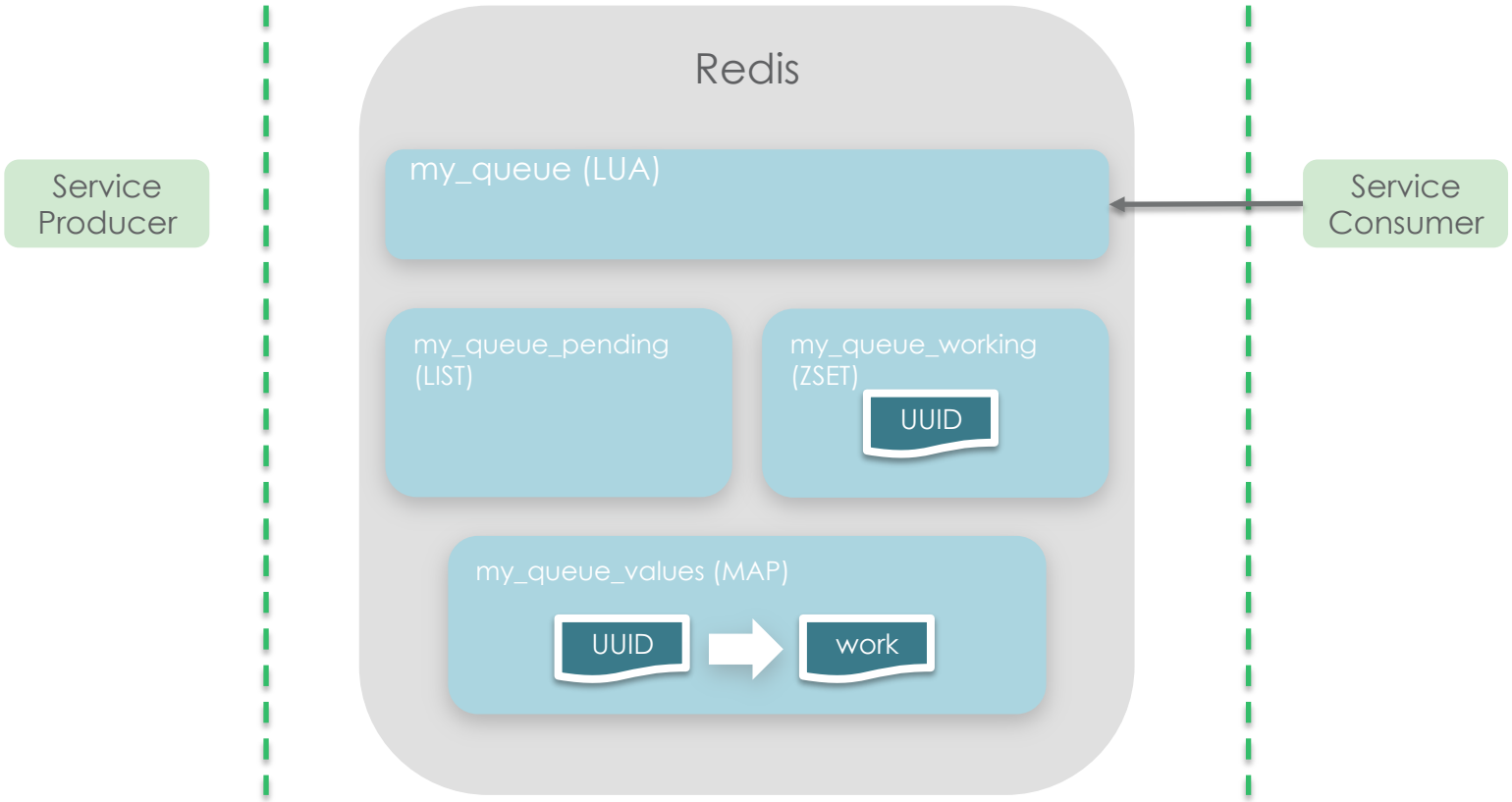
What if something goes wrong during processing?



Reliable Queueing

Bronto

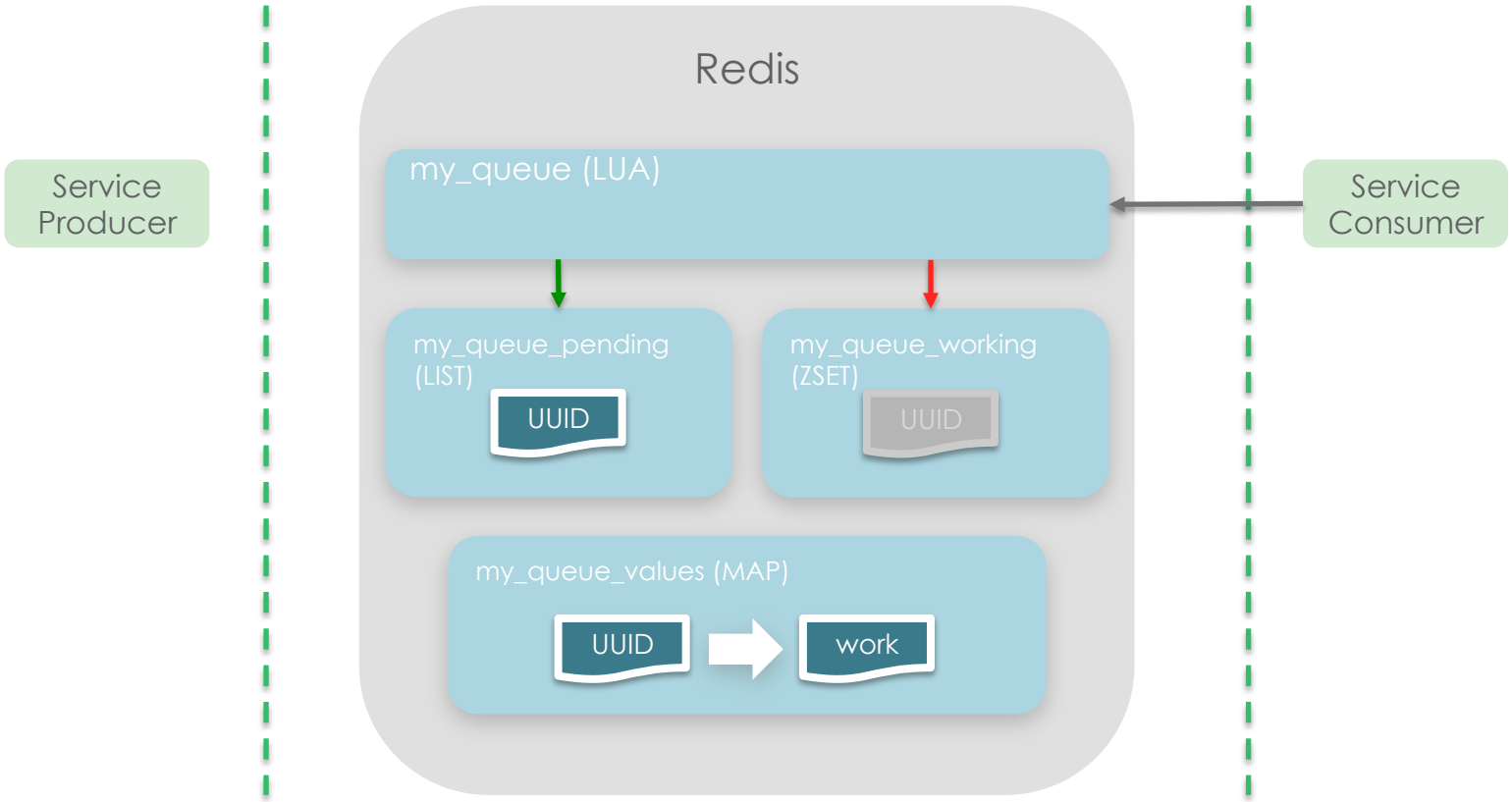
requeue()



Reliable Queueing

Bronto

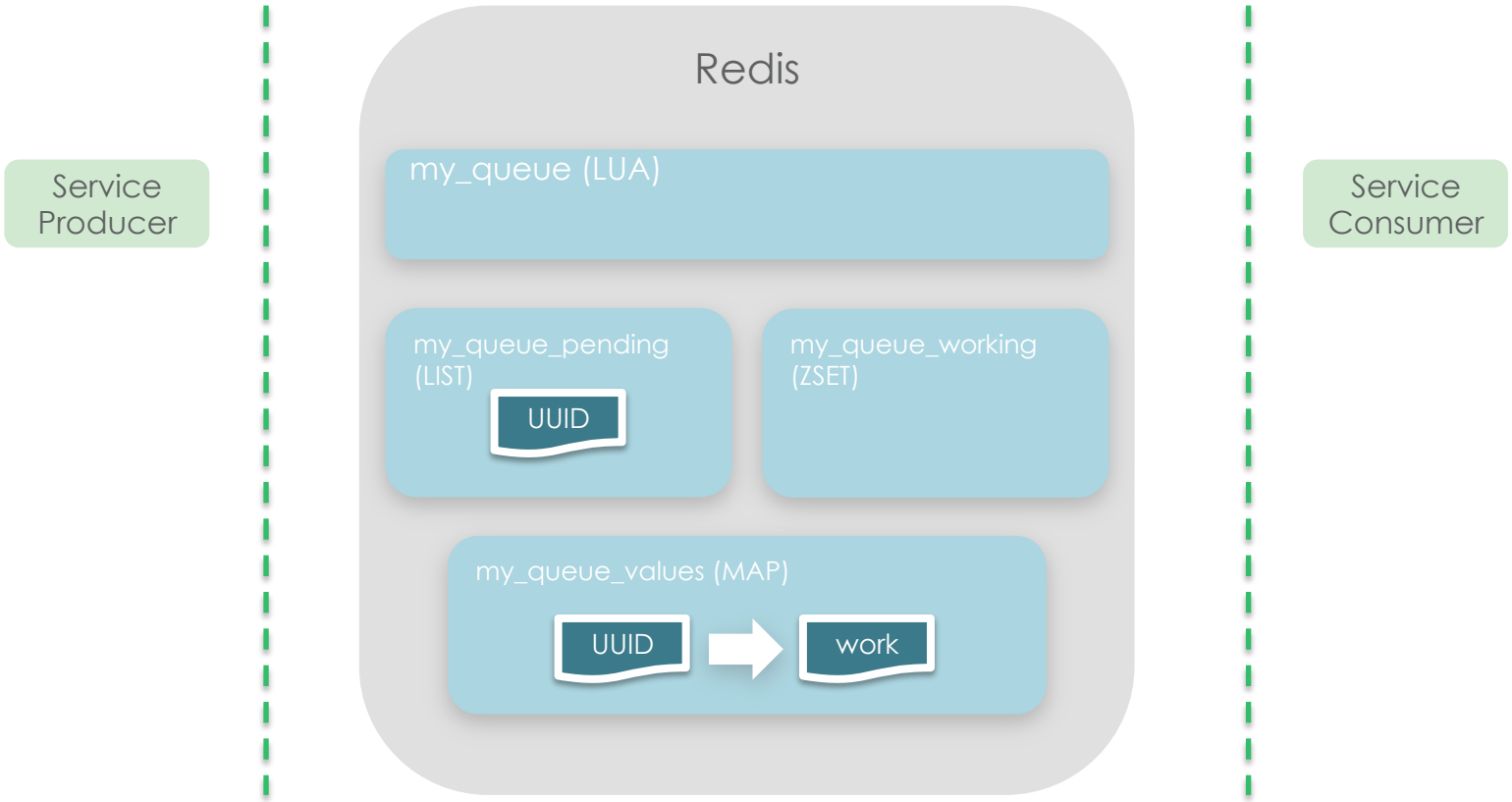
requeue()
1. ZREM {UUID}
2. LPUSH {UUID}



Reliable Queueing

Bronto

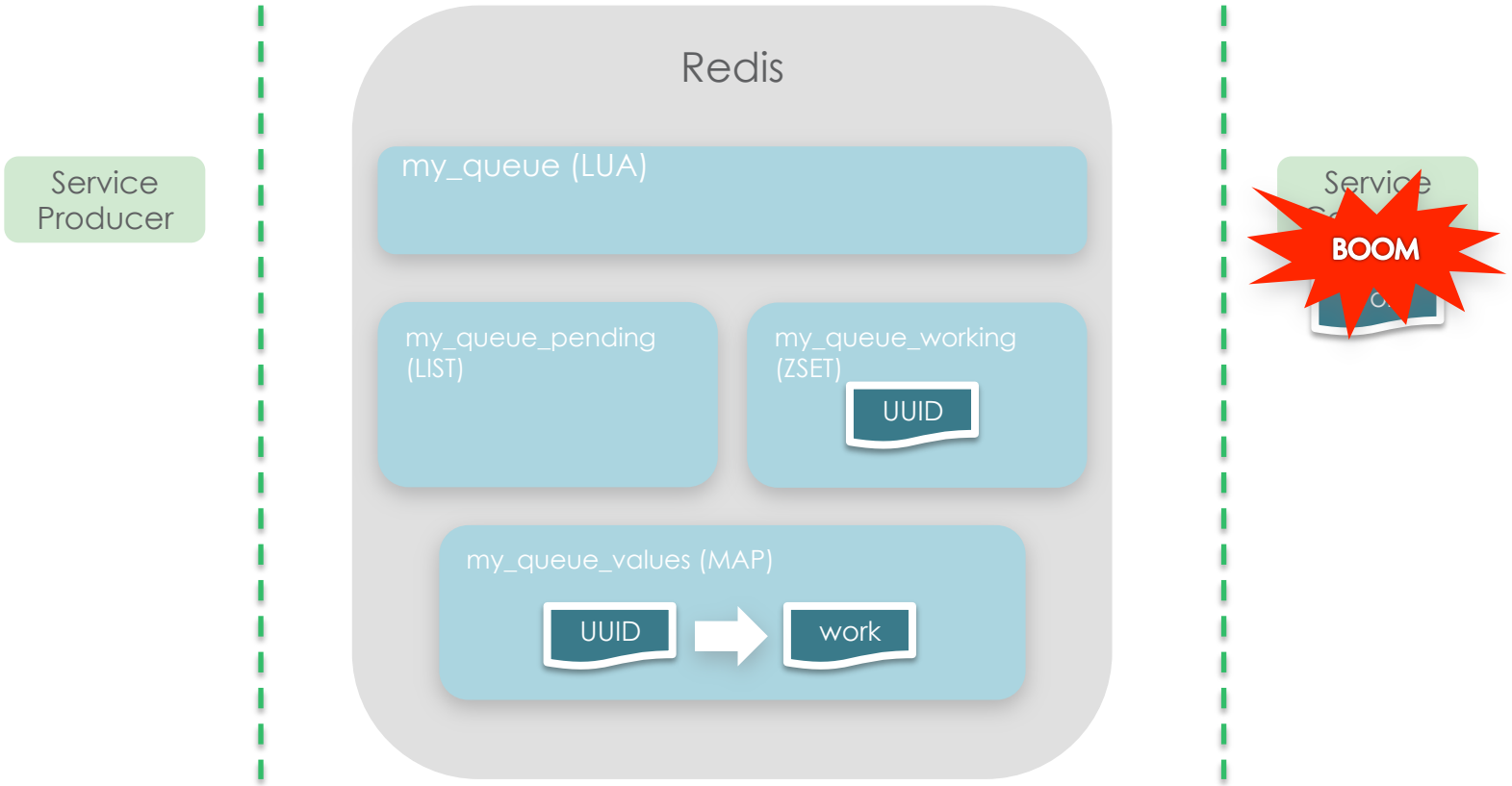
The work has now returned to the *pending* state, and will be reissued as soon as it returns to the head of the queue.



Reliable Queueing

Bronto

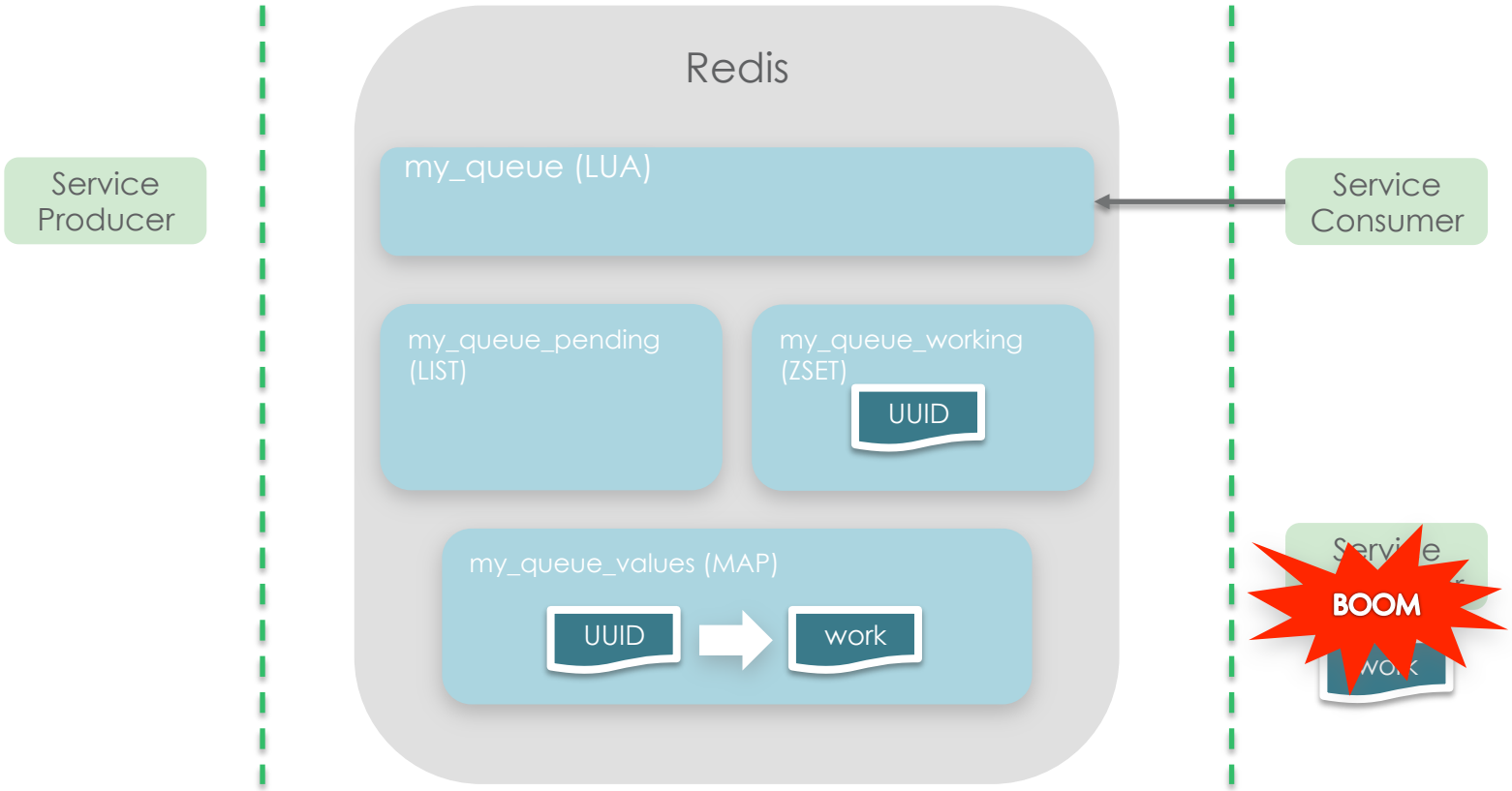
What if something *really* goes wrong during processing?



Reliable Queueing

Bronto

sweep()

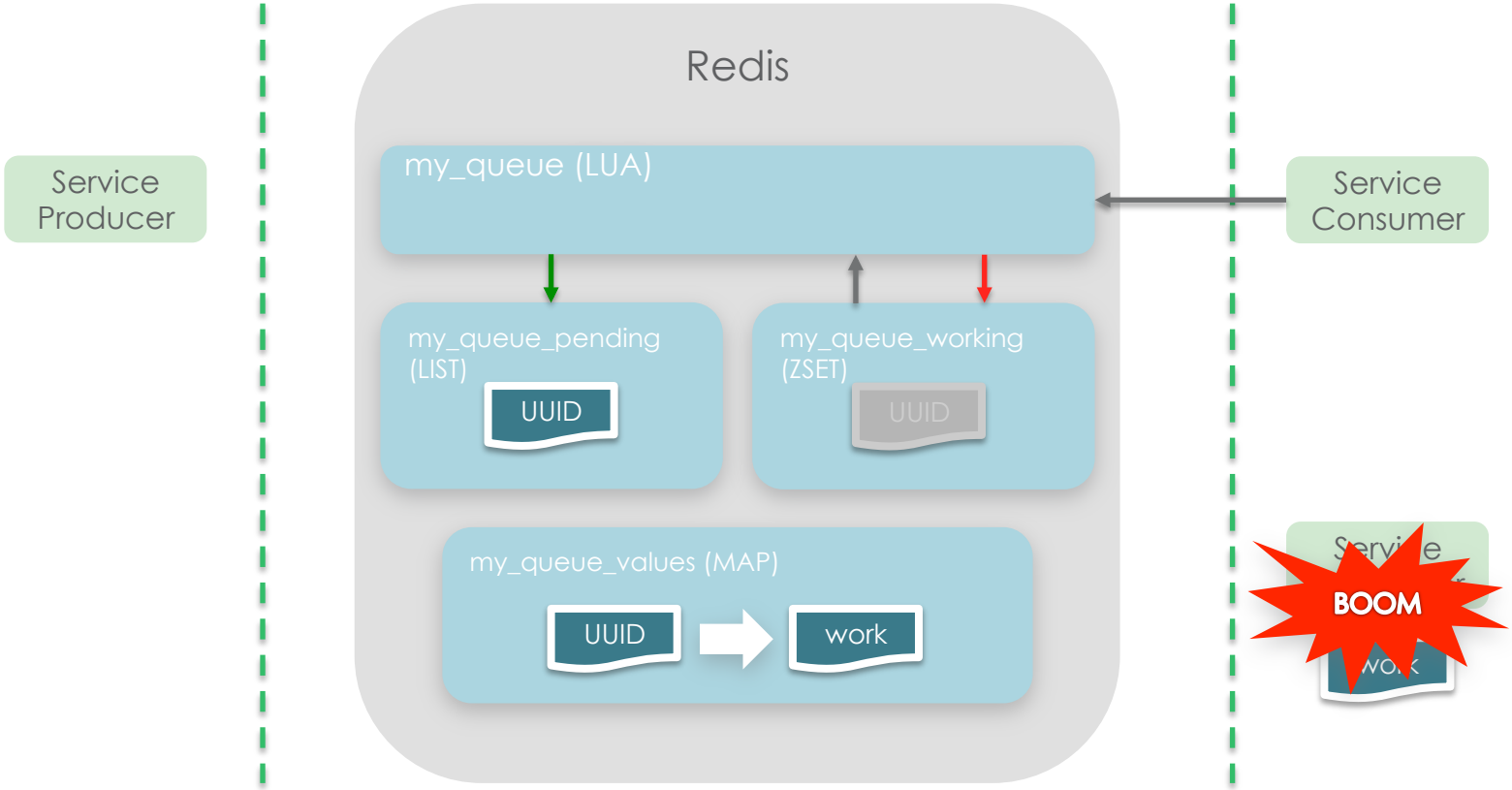


Reliable Queueing

Bronto

sweep()

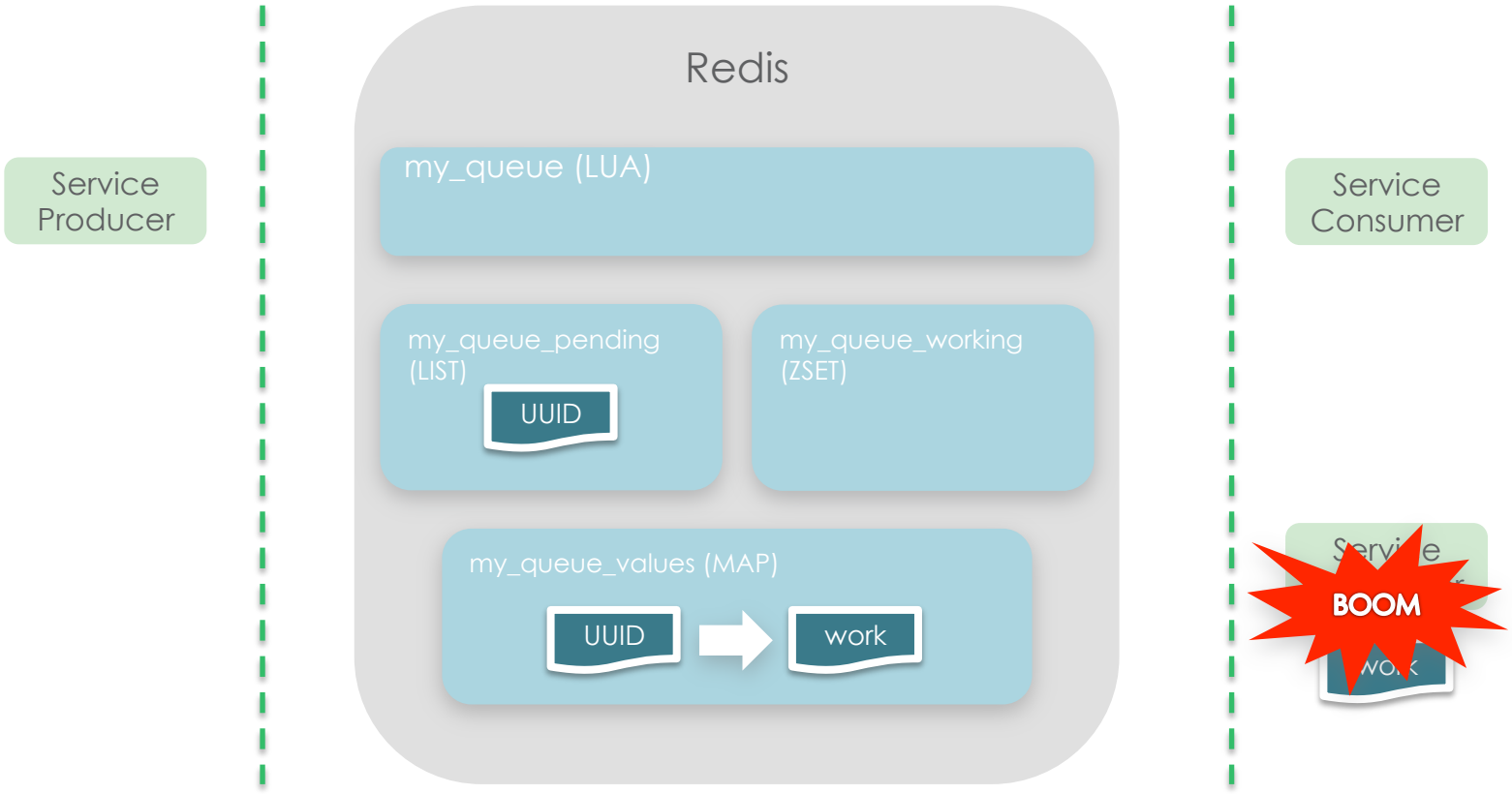
1. ZRANGEBYSCORE my_queue_working 0 {timestamp – stale}
2. LPUSH my_queue_pending {UUIDs}
3. ZREM {UUIDs}



Reliable Queueing

Bronto

The work has now returned to the *pending* state, and will be reissued as soon as it returns to the head of the queue.



Reliable Queueing

Bronto

The real implementation does a lot more…

- Asynchronous API
- Operation pipelining
- Opportunistic batching
- Pre-fetching
- Per-item deferment
- Per-item statistics
  - Enqueue time
  - Dequeue time / count
  - Requeue time / count
- Metrics instrumentation
  - Queue throughput & timing
  - Batching effectiveness
  - Queue size
  - Queue lag

Reliable Queueing

Bronto

Some benchmarks…

- Bronto's Redis Client implementation
- Bronto's Reliable Queue implementation
- Redis running on Intel(R) Xeon(R) CPU E5-2430 @ 2.20GHz
- All tests are single threaded, with one connection
- All tests use single byte queue name and item payload

Scenario	Enqueue	Dequeue & Release
No pipelining, No batching	6,700 items/sec	2,900 items/sec
Pipelining (1024), No batching	62,040 items/sec	14,029 items/sec
Pipelining (1024), Batching (256)	236,922 items/sec	70,706 items/sec

Bronto Open Source

We are planning on releasing the entire suite to the open source community.

- Redis Client
  - Asynchronous
  - Pipelining
  - Protocol access
  - Scripting supports
- Redis Benchmarking Tools
  - Scriptable benchmark runs
  - Support for rapid LUA iteration and testing
- Bronto's Reliable Queue implementation
  - Everything you just heard about

Bronto Open Source

We are planning on releasing the entire suite to the open source community.

- Redis Client
  - Asynchronous
  - Pipelining
  - Protocol access
  - Scripting supports
- Redis Benchmarking Tools
  - Scriptable benchmark runs
  - Support for rapid LUA iteration and testing
- Bronto's Reliable Queue implementation
  - Everything you just heard about

Coming this Spring.

Thanks for listening!

Bronto

Questions?